

Flooja: Visualisierung von FlockLab-Testbed-Daten

Semesterarbeit

Andreas Büchel

3. Dezember, 2014

Betreuer: Roman Lim, Marco Zimmerling, Prof. Lothar Thiele

Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich

Abstract

FlockLab ist eine Testumgebung für „Wireless Embedded“-Systeme, mit deren Hilfe mehrere verteilte Knoten zeitsynchron überwacht und angesteuert werden können. Zur Zeit gibt es keinen etablierten, standardmässigen Weg, die dabei anfallenden Daten zu visualisieren.

Cooja ist eine Simulationssoftware für „Wireless Embedded“-Systeme. Darin enthalten sind auch mehrere graphische Komponenten, *Plugins*, zur Visualisierung verschiedener Ereignisse, welche während einer Simulation auftreten z.B. das Versenden von Daten zwischen den einzelnen Knoten.

In dieser Arbeit wurde ein neues Plugin für Cooja entwickelt, genannt „Flooja“, welches die FlockLab-Daten darstellt und deren visuelle Inspektion auf benutzerfreundliche Weise erlaubt.

Danksagung

Ich möchte mich bei meinen beiden Betreuern Roman Lim and Marco Zimmerling für deren Unterstützung und hilfreiches Feedback während meiner Arbeit bedanken.

Inhaltsverzeichnis

1	Einleitung.....	5
1.1	„Wireless Embedded“-Systeme.....	5
1.2	FlockLab.....	5
1.3	Cooja.....	6
1.4	Motivation und Ziel der Arbeit.....	8
1.5	Gliederung dieses Dokuments.....	9
2	Hintergrund.....	10
2.1	FlockLab-Testlauf-Daten.....	10
2.2	Cooja und seine Plugin-Architektur.....	11
2.2.1	Simulation data flow.....	12
2.2.2	Plugins.....	12
3	Anforderungen.....	13
4	Design und Implementierung.....	14
4.1	Import der FlockLab-Testlaufdaten	14
4.2	Datenstruktur.....	14
4.2.1	GPIO-Pintracing und -Actuation.....	14
4.2.2	Powerprofiling.....	15
4.2.3	Serial output.....	16
4.3	Cooja plugin vs. Standalone-Applikation.....	16
4.4	GUI.....	16
4.4.1	Serial Output.....	17
4.4.2	FlockLab Timeline.....	18
5	Zusammenfassung und Ausblick.....	20
6	Quellen.....	21

1 Einleitung

1.1 „Wireless Embedded“-Systeme

„Wireless Embedded“-Systeme sind Systeme von verteilten, ressourcenbeschränkten, meist batteriebetriebenen Geräten, welche kabellos kommunizieren. Ein Beispiel sind kabellose Sensornetzwerke, in welchen mehrere eingebettete, mit Sensoren ausgestattete Geräte, die *Sensornodes*, örtlich verteilt Messungen an der Umgebung vornehmen und die erzeugten Daten kabellos von Node zu Node senden. Im Falle des Projektes PermaSense [1] werden mit Hilfe solcher Sensornetze z. B. verschiedene Parameter von Permafrostboden in den Schweizer Alpen ausgemessen.

1.2 FlockLab

FlockLab ist eine Testumgebung für „Wireless Embedded“-Systeme, welche am Institut für Technische Informatik der ETH Zürich entwickelt wurde [2]. FlockLab ermöglicht die zeitsynchrone Überwachung und Aktuierung mehrerer *Targets*. Diese sind dabei an räumlich verteilten Beobachterplattformen, den *Observern*, angebracht (siehe Abb. 1) und werden durch diese überwacht und angesteuert. Eine FlockLab-Installation besteht zusätzlich aus einem Server, auf welchen über ein Webinterface zugegriffen werden kann. Durch ihn werden Testläufe gestartet und die zugehörigen Resultate gesammelt und abgespeichert.

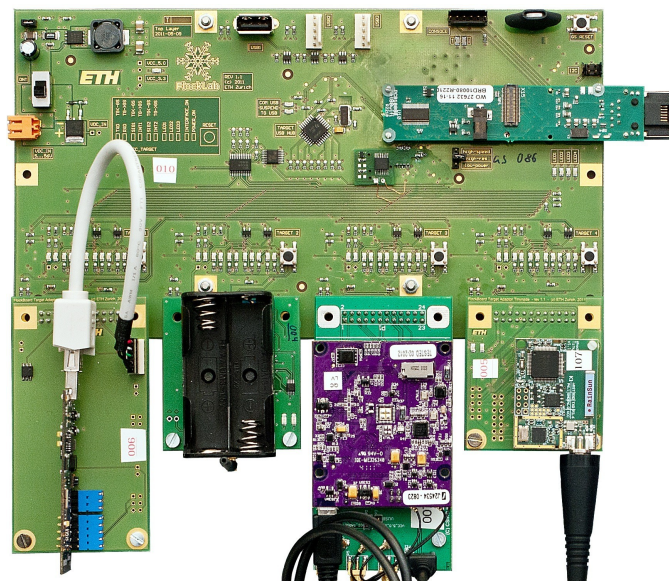


Abbildung 1: Observer mit vier Targets

Während eines Testlaufs wird auf dem Target das zu testende Softwaresystem ausgeführt. Durch den Observer können verschiedene Aspekte des Targets beeinflusst bzw. überwacht werden. Die Hauptfunktionen sind dabei *GPIO-Tracing*, *GPIO-Actuation*, Messung der Stromaufnahme, Kontrolle über die Versorgungsspannung und schliesslich das Loggen der Nachrichten, welche über die serielle Schnittstelle des Targets gesendet bzw. empfangen werden. Im Folgenden werden die einzelnen Funktionen genauer beschrieben.

GPIO-Pin-Tracing

Jedes Target verfügt über einige GPIO-Pins. Ereignisse auf diesen, also steigende bzw. fallende Flanken, werden vom Observer erkannt und zusammen mit der Zeit des Auftretens gespeichert. Werden diese Pins durch die auf dem Target laufende Software gesetzt, so kann von diesen Änderungen der Pins mit hoher zeitlicher Auflösung auf Zustandsänderungen im Target zurückgeschlossen werden. Durch Setzen eines Pins auf "high" bzw. "low" vor bzw. nach einer Datenübertragung über das Funkmodul des Targets zum Beispiel, können die Zeitabschnitte, in denen gesendet wurde, nachverfolgt werden.

GPIO-Pin-Actuation

Einige der GPIO-Pins des Targets können vom Observer auf ein bestimmtes Niveau gesetzt werden. So können z.B. die Targets am Anfang eines Testlaufs durch Setzen des Reset-Pins neu gestartet werden.

Powerprofiling

Die Stromaufnahme des Targets kann mit einer Samplingfrequenz bis zu 56 kHz gemessen werden.

Setzen der Versorgungsspannung

Im Verlauf eines Testlaufs kann die Versorgungsspannung des Targets auf bestimmte Werte gesetzt werden. Dies ermöglicht z.B. die Simulation einer Batterieentladung.

Serial I/O

Über die serielle Schnittstelle des Targets können Nachrichten gesendet bzw. empfangen werden. Diese Kommunikation wird durch den Observer geloggt, d.h. Informationen darüber, welche Nachricht zu welcher Zeit in welche Richtung gesendet wurde, werden abgespeichert.

1.3 Cooja

Cooja ist ein Softwaresimulator für „Wireless Embedded“-Systeme. Mit Cooja können mehrere Sensorknoten virtuell räumlich angeordnet, ein Programm darauf ausgeführt und der Zustand der Knoten währenddessen beobachtet werden. So kann für jeden Knoten z.B. nachverfolgt werden, wann er welche Daten über sein Funkmodul sendet bzw. empfängt. Für die zu simulierenden Sensorknoten werden dazu entsprechende Hardwaresimulatoren verwendet.

Abb. 2 zeigt Coojas graphische Oberfläche bestehend aus mehreren Fenstern (*Plugins*), in welchen verschiedene Ereignisse und Parameter der Simulation auf verschiedene Art und Weise dargestellt werden. Während einige dieser Plugins standardmässig in Cooja enthalten sind, ist es möglich eigene zu programmieren und zu Cooja hinzuzufügen. Zwei der Standardplugins werden im Weiteren genauer vorgestellt, da die darin angezeigten Daten ähnlich denjenigen Daten sind, welche bei einem FlockLab-Testlauf generiert werden.

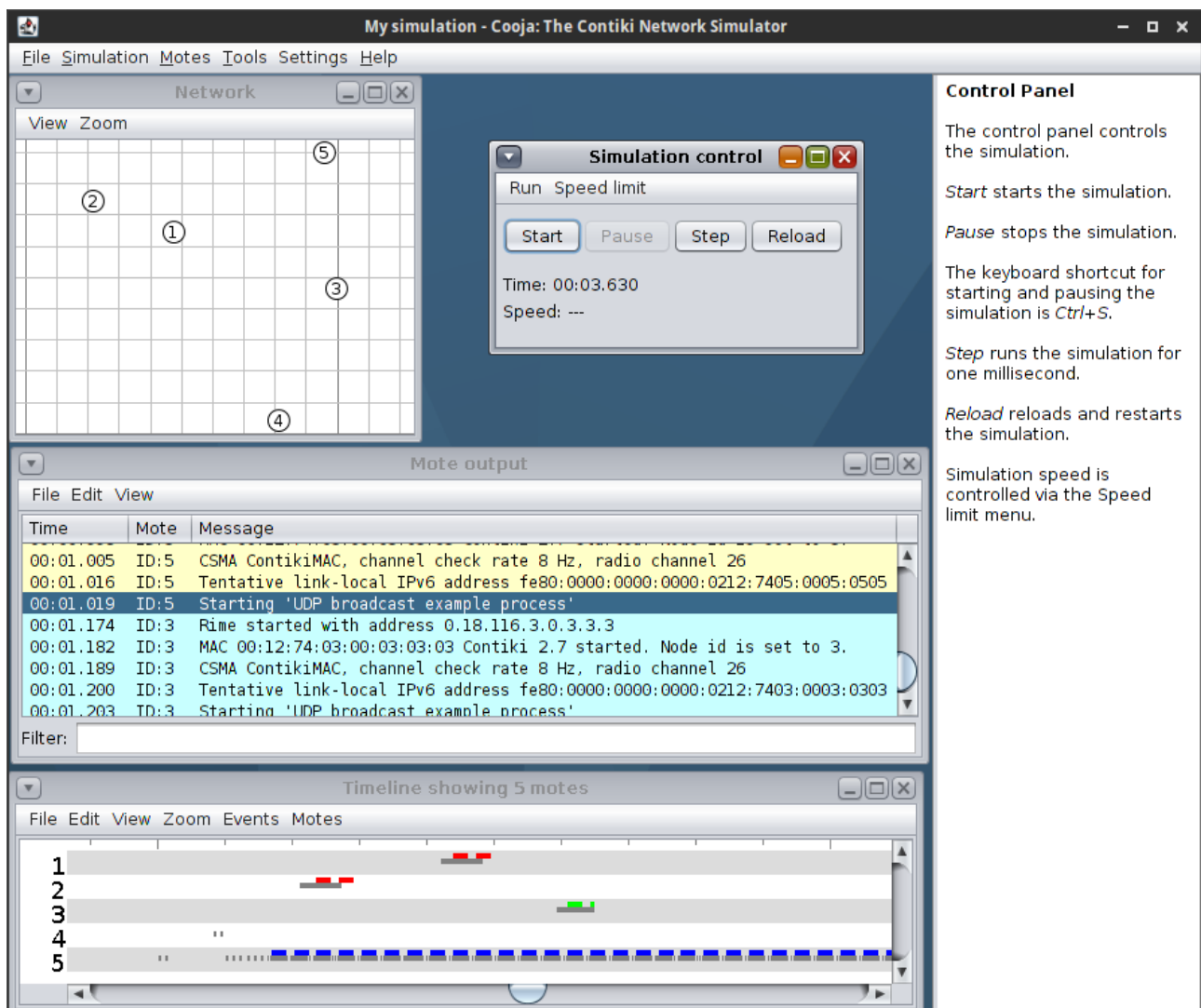


Abbildung 2: Cooja mit Plugins

„Timeline“

Im „Timeline“-Plugin (Abb. 3) werden verschiedene Zustände der simulierten Knoten im zeitlichen Verlauf dargestellt. Unter anderem ist darin ersichtlich, wann das Funkmodul eines Knotens ein- bzw. ausgeschaltet ist (graue Balken) und wann er damit Daten sendet (blau), empfängt (grün) bzw. durch einen anderen Knoten gestört wird (rot).

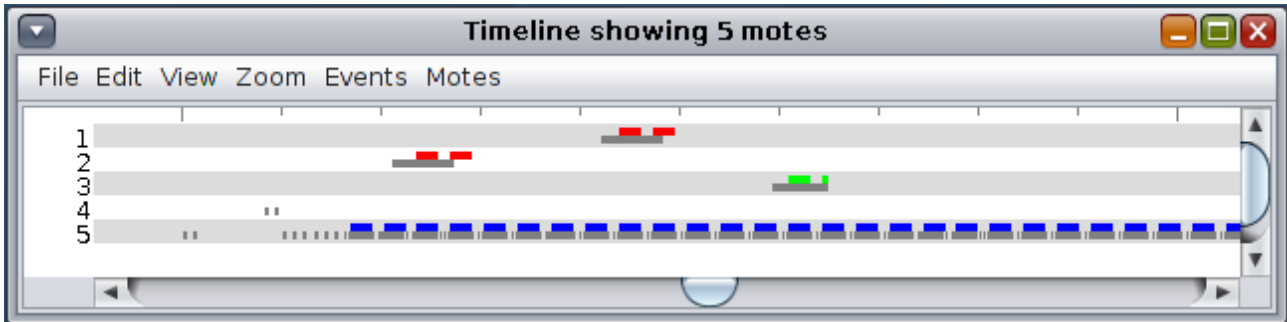


Abbildung 3: Coojas „Timeline“-Plugin

„Mote output“

Im „Mote output“-Plugin werden Nachrichten dargestellt, welche von den Knoten verschickt werden. Die in Form einer Tabelle dargestellten Nachrichten (siehe Abb. 4) können mit Hilfe von regulären Ausdrücken anhand der ID (Spalte "Mote") und der eigentlichen Nachricht (Spalte "Message") gefiltert werden.

The screenshot shows a window titled "Mote output" with a menu bar "File Edit View". It contains a table with three columns: "Time", "Mote", and "Message". The table lists messages from five nodes, color-coded by node ID. At the bottom, there is a "Filter:" input field.

Time	Mote	Message
00:00.511	ID:2	Rime started with address 0.18.116.2.0.2.2.2
00:00.519	ID:2	MAC 00:12:74:02:00:02:02:02 Contiki 2.7 started. Node id is set to 2.
00:00.526	ID:2	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.537	ID:2	Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7402:0002:0202
00:00.540	ID:2	Starting 'UDP broadcast example process'
00:00.623	ID:4	Rime started with address 0.18.116.4.0.4.4.4
00:00.631	ID:4	MAC 00:12:74:04:00:04:04:04 Contiki 2.7 started. Node id is set to 4.
00:00.638	ID:4	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.649	ID:4	Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7404:0004:0404
00:00.652	ID:4	Starting 'UDP broadcast example process'
00:00.657	ID:1	Rime started with address 0.18.116.1.0.1.1.1
00:00.665	ID:1	MAC 00:12:74:01:00:01:01:01 Contiki 2.7 started. Node id is set to 1.
00:00.672	ID:1	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.683	ID:1	Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
00:00.686	ID:1	Starting 'UDP broadcast example process'
00:00.990	ID:5	Rime started with address 0.18.116.5.0.5.5.5
00:00.998	ID:5	MAC 00:12:74:05:00:05:05:05 Contiki 2.7 started. Node id is set to 5.
00:01.005	ID:5	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:01.016	ID:5	Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7405:0005:0505
00:01.019	ID:5	Starting 'UDP broadcast example process'

Abbildung 4: Coojas „Mote output“-Plugin

1.4 Motivation und Ziel der Arbeit

Zur Zeit gibt es keinen Standardweg, die Daten, welche bei einem Testlauf von FlockLab generiert wurden, zu visualisieren. Es ist dem Benutzer von FlockLab überlassen, wie er die Daten visualisieren möchte. Dafür wird z.B. Matlab verwendet.

Während dieser Arbeit soll dieses Problem angegangen werden, indem eine öffentlich zugängliche Software erstellt wird, welche diese Visualisierung auf einfache Weise erlaubt. Es soll ein Plugin für Cooja entwickelt werden, welches die Resultate eines FlockLab-Testlaufs auf benutzerfreundliche Art und Weise darstellt und es erlaubt, diese komfortabel zu inspizieren. Dabei soll als Grundlage auf den bereits in Cooja vorhandenen Plugins „Timeline“ und „Mote output“ aufgebaut werden. Insbesondere sollen auch die Resultate der Strommessung von FlockLab mit Hilfe

dieses Plugins ähnlich Abb. 5 dargestellt werden.

Im Folgenden wird dieses Plugin „Flooja“ genannt.

Die vollständige, offizielle Aufgabenstellung ist am Ende dieses Dokumentes angefügt.

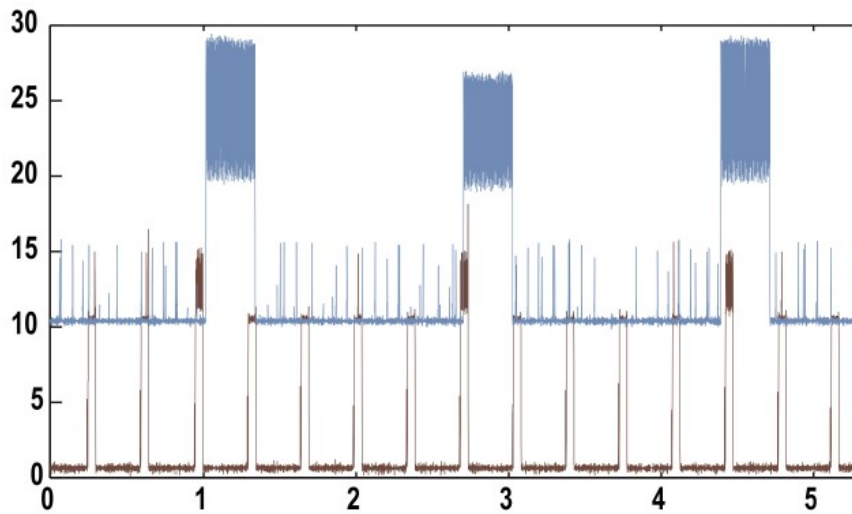


Abbildung 5: Beispiel einer Visualisierung zweier Strommessungen

1.5 Gliederung dieses Dokumentes

Nach der Einleitung in Kapitel 1 werden in Kapitel 2 die von FlockLab generierten Daten sowie Cooja genauer beschrieben. Kapitel 3 listet die Anforderungen für Flooja auf, welche im Verlauf dieser Arbeit festgelegt wurden. In Kap. 4 wird anschliessend auf das Design und die Implementierung von Flooja eingegangen. Dabei wird zum Einen beschrieben, wie die Daten in Flooja geladen, intern verwaltet und schliesslich angezeigt werden, und zum Anderen die graphische Benutzeroberfläche von Flooja erläutert. Abschliessend finden sich in Kap. 5 eine Zusammenfassung dieser sowie ein Ausblick auf mögliche weitere Arbeiten an Flooja.

2 Hintergrund

2.1 FlockLab-Testlauf-Daten

In diesem Abschnitt werden die bei einem FlockLab-Testlauf anfallenden Daten beschrieben.

Über das Webinterface von FlockLab können die Resultate eines FlockLab-Testlaufs heruntergeladen werden. Die verschiedenen Messergebnisse und Ereigniss-Logs sind dabei in mehrerer CSV-Dateien hinterlegt, welche in Form einer einzigen komprimierten Archivdatei (tar.gz) heruntergeladen werden können.

Dabei handelt es sich um die folgenden Dateien:

- errorlog.csv
- gpiotracing.csv
- powerprofilingstats.csv
- gpioactuation.csv
- powerprofiling.csv
- serial.csv

Jede Datei enthält eine Tabelle im CSV-Format bestehend aus mehreren Zeilen und Spalten. Ausgenommen *powerprofilingstats.csv* entspricht dabei jede Zeile einem zeitlichen Ereignis z.B. einer Messung der Stromaufnahme zu einer bestimmten Zeit. Jedes dieser Ereignisse wurde bei einem bestimmten Observer-Target-Paar registriert. Aus diesem Grund enthalten diese CSV-Dateien die folgenden Spalten:

- timestamp: Zeit des Ereignisses gemessen in Sekunden seit dem 1. Jan 1970 00:00.
- observer_id: ID des Observers, auf welchem das Ereignis registriert wurde
- target_id: ID des Targets, für welches das Ereignis registriert wurde

Jede CSV-Datei enthält zusätzliche Spalten mit weiteren, das jeweilige Ereignis betreffenden Informationen. Die einzelnen Dateien werden im Folgenden genauer beschrieben.

errorlog.csv enthält Lognachrichten bezüglich während des Testlaufs aufgetretener Fehler. In der Spalte „errormessage“ befindet sich die entsprechende Fehlermeldung.

gpiotracing.csv enthält Ereignisse, welche an einem GPIO-Pin des Targets aufgetreten sind. Die Spalte „pin_name“ enthält den Namen des Pins, die Spalte „value“ entweder 0 oder 1 entsprechend einer negativen oder positiven Flanke. Abb. 6 zeigt den Inhalt von *gpiotracing.csv*.

```
# timestamp,observer_id,node_id,pin_name,value
1410958440.335820,19,19,INT1,1
1410958440.335893,19,19,LED3,1
1410958440.335901,19,19,LED2,1
1410958440.335907,19,19,INT2,1
1410958441.403649,19,19,INT1,0
1410958441.403842,19,19,LED3,0
1410958441.403850,19,19,INT2,0
1410958441.415366,19,19,INT1,1
1410958441.481108,19,19,INT1,0
```

Abbildung 6: *gpiotracing.csv*

gpioactuation.csv (Abb. 7) enthält Informationen über Aktuierungen von GPIO-Pins bestimmter Targets durch den Observer. Diese Datei enthält, im Gegensatz zu den anderen, zwei Spalten mit Zeitinformation. „timestamp_planned“ gibt den Zeitpunkt an, zu welchem die Aktuierung des Pins auf dem Observer geplant wurde; „timestamp_executed“ denjenigen der tatsächlichen Ausführung. Wie in *gpiotracing.csv* enthält auch diese Datei die Spalten „pin_name“ und „value“ mit der selben Bedeutung.

```
# timestamp_planned,timestamp_executed,observer_id,node_id,pin_name,value
1410958440.000000,1410958440.000057,13,13,RST,0
1410958800.000000,1410958800.000057,13,13,RST,1
1410958440.000000,1410958440.000066,27,27,RST,0
1410958800.000000,1410958800.000064,27,27,RST,1
1410958440.000000,1410958440.000189,10,10,RST,0
1410958800.000000,1410958800.000063,10,10,RST,1
```

Abbildung 7: *gpioactuation.csv*

powerprofiling.csv (Abb. 8) enthält Messwerte der Stromaufnahme durch die Targets zur entsprechenden Zeit. Die Spalte „*value_mA*“ enthält die gemessenen Werte in Milliampere.

```
# timestamp,observer_id,node_id,value_mA
1410958500.035737,23,23,136.131215
1410958500.035806,23,23,23.526857
1410958500.035876,23,23,23.491518
1410958500.036293,23,23,23.495980
1410958500.678417,1,1,24.412981
1410958500.678487,1,1,24.560918
1410958500.678556,1,1,24.209704
1410958500.678695,1,1,24.249319
1410958500.678765,1,1,24.226404
```

Abbildung 8: *powerprofiling.csv*

powerprofilingstats.csv (Abb. 9) enthält für jedes Target genau einen Eintrag bestehend aus dem zeitlichen Mittelwert der Stromaufnahme im Milliampere (Spalte „*mean_mA*“). Diese Datei enthält keine Zeitstempel.

```
# observer_id,node_id,mean_mA
1,1,24.339049
2,2,24.234394
3,3,0.028450
4,4,24.461108
```

Abbildung 9: *powerprofilingstats.csv*

serial.csv (Abb. 10) enthält Nachrichten (Spalte „*output*“), welche über die serielle Schnittstelle des entsprechenden Targets empfangen bzw. gesendet wurden. Die Spalte „*direction*“ gibt dabei die Richtung der Kommunikation an.

```
# timestamp,observer_id,node_id,direction,output
1410958440.332164,13,13,r,Contiki 2.4 started. Node id is set to 13.
1410958440.332164,13,13,r,Starting 'Glossy test'
1410958440.394009,13,13,r,glossy test! tx power: 31, proc cycles: 1500...
1410958462.500729,13,13,r,noisefloor -91 -90
1410958462.500729,13,13,r,seq_no 4
1410958462.500729,13,13,r,skew -16
```

Abbildung 10: *serial.csv*

2.2 Cooja und seine Plugin-Architektur

Im Folgenden soll Coojas interner Aufbau erläutert und damit der Hintergrund für die Ausführungen im Kapitel “Design and Implementation” geschaffen werden.

2.2.1 Simulation data flow

Cooja ist intern als ereignisorientierter Simulator implementiert. Für verschiedene Werte, welche sich während der Simulation verändern (z.B. die Menge der Knoten welche gerade vorhanden sind oder – spezifischer – der Zustand einer LED bei einem bestimmten Node) kann bei der Simulation ein *Listener*-Objekt registrieren werden, welches bei einer Änderung des entsprechenden Wertes darüber informiert wird.

In der Simulation selbst werden die einzelnen Ereignisse nur soweit behandelt, dass alle interessierten Listener beim

Auftreten darüber informiert werden. Die Ereignisse werden jedoch nicht für späteren Zugriff abgespeichert.

So registriert sich beispielsweise das bereits in Cooja vorhandene „Timeline“-Plugin als Listener für Änderungen an den LED-Werten, dem Zustand des Radiomoduls (on/off, receiving/transmitting/idle) und dem Logoutput der simulierten Knoten. Um die Ereignisdaten jedoch für die Darstellung länger zur Verfügung zu haben, werden sie hier zwischengespeichert.

2.2.2 Plugins

Wie bereits beschrieben, besteht die graphische Oberfläche von Cooja aus mehreren Fenstern, genannt Plugins. Zum einen werden in diesen verschiedene Aspekte von Cooja dargestellt z.B. die räumliche Verteilung der simulierten Knoten oder der momentane Wert bestimmter Eigenschaften einzelner Knoten. Zum Anderen kann durch Plugins Einfluss auf Cooja genommen werden. Siehe z.B. das Plugin „Simulation control“ in Abb. 2, über welches die Simulation angehalten, wiederaufgenommen oder schrittweise durchgegangen werden kann.

Cooja erlaubt es, eigene Plugins zu entwickeln und diese einzubinden. Wie Cooja selbst werden auch diese mithilfe von Java und der GUI-Bibliothek „Swing“ erstellt. Ein Plugin besteht dabei aus einem Verzeichnis (in unserem Fall „flooja“) mit dem folgenden Inhalt:

- Ein Unterverzeichnis „java“. Dieses enthält die kompilierten Java-Klassen, durch welche das Plugin implementiert ist.
- Eine Textdatei „cooja.config“. Darin wird diejenige Klasse aus dem „java“-Verzeichnis angegeben, über welche das Plugin gestartet wird. Zusätzlich enthält „cooja.config“ eine kurze Beschreibung des Plugins.

Über „Settings“ → „Cooja extensions...“ können zusätzliche Plugins wie in Abb. 11 und 12 gezeigt aktiviert werden. Über das Menü „Tools“ kann das Plugin anschliessend gestartet werden (Abb. 13).

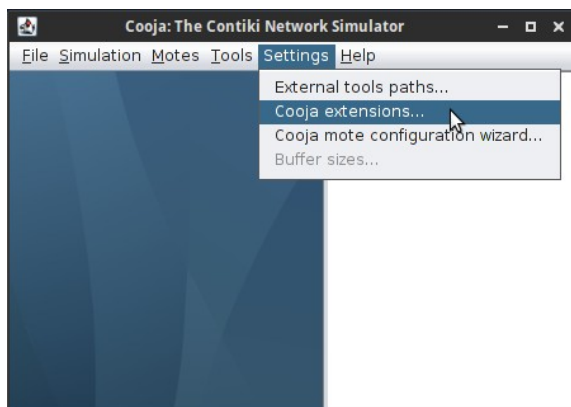


Abbildung 11: Aktivieren eines Plugins (Schritt 1)

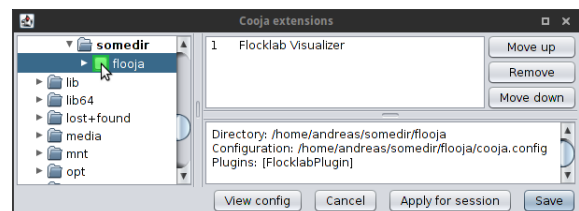


Abbildung 12: Aktivieren eines Plugins (Schritt 2)

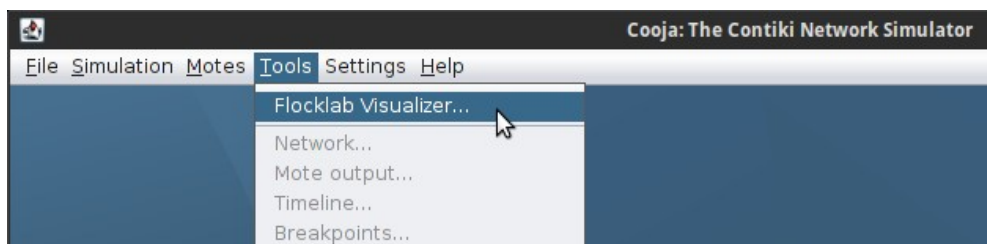


Abbildung 13: Starten von Flooja

3 Anforderungen

Im folgenden werden die Anforderungen an Flooka aufgelistet, welche im Verlauf der Arbeit festgelegt wurden.

- Visualisierung
 - "FlockLab Timeline"
 - Die Strommessungen aus *powerprofiling.csv* werden in einem Plot gegen die Zeit ähnlich Abb. 5 dargestellt.
 - GPIO-Trace- und -Actuation-Daten werden ähnlich zum bereits in Cooja vorhandenen „Timeline“-Plugin dargestellt, wobei ersichtlich ist, in welchen Zeitabschnitten der entsprechende Pin „high“ bzw. „low“ war.
 - Die Nachrichten aus *serial.csv* werden ähnlich wie im bereits vorhandenen Plugin „Mote output“ tabellarisch dargestellt mit je einer Spalte für Zeitstempel, Observer-ID, Target-ID, Richtung der Kommunikation und Nachricht selbst.
- Benutzerfreundliche Inspektion der Daten
 - Die Nachrichten aus *serial.csv* können nach Target und Inhalt gefiltert werden.
 - Die Skala der Zeitachse in der „FlockLab Timeline“-Komponente kann verändert werden (Zooming).
 - Es ist möglich, durch Markieren eines bestimmten Zeitpunktes in einer Komponente (Tabelle mit Nachrichten oder „FlockLab Timeline“) in der jeweils anderen Komponente zum selben Zeitpunkt zu springen.
 - Für jeden GPIO-Pin kann durch den Benutzer ein kurzer, beschreibender Text definiert werden, welcher an geeigneter Stelle angezeigt wird.
 - Vom Benutzer veränderbare Einstellungen wie z.B. die Pin-Beschreibungen können in eine Konfigurationsdatei abgespeichert und daraus wiederhergestellt werden.

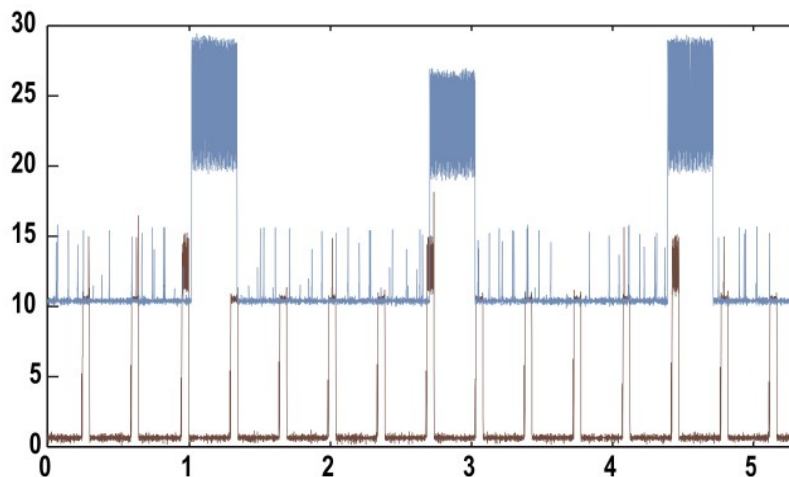


Abbildung 14: Visualisierung der Stromaufnahme zweier Targets

4 Design und Implementierung

4.1 Import der FlockLab-Testlaufdaten ...

Zunächst soll hier der gewählte Weg erläutert werden, über den die Daten eines FlockLab-Testlaufs in *Flooja* hinein und schliesslich zu den darstellenden GUI-Elementen gelangen. Zwei Optionen dafür werden im Folgenden erläutert, wovon die zweite schliesslich gewählt wurde.

... via Cooja-Simulation

Um so viele der bereits vorhandene Komponenten von *Cooja* wie möglich wiederzuverwenden, würde der vielleicht offensichtlichste Ansatz zum Laden der FlockLab-Daten irgendwie über die Simulation führen. Die Simulation ist in Cooja jedoch konzipiert als Komponente, welche Ereignisse gemäss ihrer zeitlichen Reihenfolge abarbeitet, andere Komponenten jeweils darüber informiert, danach jedoch die bereits behandelten Ereignisse nicht weiter abspeichert.

In einem eigenen Design könnte nun die Simulation so abgeändert werden, dass sie die auftretenden Ereignisse nicht durch einen Hardwaresimulator der verwendeten Knoten generiert, sondern einfach aus den Resultaten des FlockLab-Testlaufs liest. Die FlockLab-Daten würden in zeitlicher Reihenfolge abgearbeitet und entsprechende Ereignisse generiert, für deren Behandlung sich die graphischen Komponenten zuvor registriert hätten. Dieser Ansatz wurde jedoch aufgrund der folgenden Probleme nicht weiter verfolgt.

Zum einen sollte ein Plugin für Cooja entwickelt werden. Die Simulationskomponente von *Cooja* lässt sich durch Plugins jedoch nicht verändern. Zum Anderen kann die Menge der Daten, welche bei einem FlockLab-Testlauf generiert werden, ziemlich gross sein; vor allem bei der Messung der Stromaufnahme eines Targets. Für jedes Sample fallen dabei der Zeitstempel (64-bit-Integer) und ein Messwert (64-bit-Float) an. Bei einer nicht unrealistischen Testdauer von 60min und einer Samplingfrequenz von 14.4 kHz macht das ca. 790 MB pro Target; bei 30 Targets also insgesamt 23 GB. Da die Simulation selbst die Daten nur durchgehen und entsprechende Ereignisse generieren würde, müssten sie in den GUI-Komponenten zwischengespeichert werden. Aufgrund der grossen Menge an Daten wäre dafür der Arbeitsspeicher jedoch nicht ausreichend und sie müssten auf die Festplatte ausgelagert werden. Von der Festplatte selbst liest jedoch die Simulation schon alle Daten.

... direkt durch das Plugin

Nach den obigen Überlegungen wurde für *Flooja* ein anderer Weg gewählt, um die FlockLab-Testlaufdaten zu laden. Die Simulation wird umgangen und die darzustellenden Daten werden direkt vom Plugin aus den Dateien geladen. Dies ist vor allem hilfreich bei der Darstellung der Strommessungen, denn der Zugriff auf die Teile der Messdaten, welche gerade dargestellt werden sollen, kann nun einfach durch Zugriff auf Teile der entsprechenden Dateien geschehen.

Wird *Flooja* nun innerhalb von Cooja gestartet, wird der User nach dem Ordner gefragt in welchem sich die FlockLab-Testlaufdaten befinden und diese werden dann von dort geladen. Obwohl *Flooja* dadurch quasi eine allein lauffähige Software ist, welche via Cooja gestartet wird, wurde die Verbindung mit Cooja beibehalten, um der Zielgruppe des Plugins, Entwickler und Forscher im Bereich von „Wireless Embedded“-Systemen, in vertrauter Umgebung zu begegnen.

4.2 Datenstruktur

In den folgenden Abschnitten werden die Datenstrukturen erläutert, in welchen die FlockLab-Testlaufdaten von *Flooja* geladen und intern gehandhabt werden.

4.2.1 GPIO-Pintracing und -Actuation

Bis auf den Zeitstempel sind die Daten von GPIO-Tracing bzw. -Actuation identisch (siehe oben). Für *Flooja* sollte bei den Actuation-Daten nur die „time_executed“ verwendet werden, womit die Datensätze schliesslich gleich behandelt werden können.

Wie bereits beschrieben, enthalten die GPIO-Daten eines FlockLab-Testlaufs Informationen darüber, an welchem Observer-Target-Paar, zu welcher Zeit, sich welcher Pin, wie verändert hat (positive oder negative Flanke). Im Hinblick auf die spätere Visualisierung wurden diese Daten folgendermassen unterteilt. Zuerst nach dem Target. Für jedes Target, identifiziert durch seine ID (ein Integerwert), wird dann weiter nach dem Namen des betrachteten GPIO-Pins unterschieden. Für ein bestimmtes Target und einen bestimmten GPIO-Pin bleibt schliesslich eine Menge von zeitlichen Ereignissen (Zeitwert, Boolean) übrig, wobei der boolesche Wert eine steigende bzw. fallende Flanke des Pins

ausdrückt.

In *Flooja* werden diese Daten in ein Objekt der folgenden Klasse geladen:

```
HashMap<Integer,HashMap<String,TreeMap<Long,Boolean>>>
```

`HashMap` und `TreeMap` sind Klassen der Java API und bieten übliche Funktionalität einer Hash-Tabelle (Schlüssel-Wert-Paare). `TreeMap` ist jedoch dahingehend besonders, dass die Einträge zusätzlich eine Ordnung gemäss dem Schlüssel (hier Zeitstempel) haben. Des Weiteren bietet `TreeMap` Methoden zum Zugriff auf Teilbereiche der Schlüssel-Wert-Paare an. Damit kann bei der Visualisierung effizient auf die benötigten Werte innerhalb eines bestimmten Zeitintervalls zugegriffen werden. Die bereits in Java enthaltene Implementierung der `TreeMap` basiert auf einem Rot-Schwarz-Baum.

Für ein bestimmtes Target und einen seiner GPIO-Pins (gegeben durch den Pin-Namen) bezeichnen wir die zugehörige `TreeMap<Long, Boolean>` als die „GPIO-Trace“ dieses Pins.

4.2.2 Powerprofiling

Dieser Abschnitt soll erläutern, wie die Strommessdaten geladen werden und schliesslich zu den graphischen Elementen von *Flooja* gelangen.

Für die Darstellung der Strommesswerte in Form eines Plots (Zeit gegen Messwert) werden die Messwerte in einem bestimmten Zeitintervall benötigt. Für diese Aufgabe wird in *Flooja* das Interface `CurrentTrace` definiert, worüber auf die zu einem bestimmten Target gehörenden Strommesswerte zugegriffen werden kann. Es stellt im wesentlichen eine Operation zur Verfügung: `getMeasurementsCovering(long startTime, long endTime)`. Rückgabewert dieser Methode sind die Samplezeiten und die zugehörigen Messwerte zwischen `startTime` und `endTime`.

Gleiche Datenstruktur wie für GPIO-Ereignisse

In einer ersten Implementierung verwendeten wir für die Strommesswerte die selbe Datenstruktur wie für die GPIO-Trace- und -Actuation-Daten. Einziger Unterschied ist, dass zu jeder Zeit nicht ein boolescher Wert, sondern ein 64-bit-Float-Wert verwendet wird, welcher die Stromaufnahme in Milliampere des Targets zur entsprechenden Zeit angibt. Die Messzeiten und -werte in einem bestimmten Zeitintervall können wie bereits beschrieben einfach aus der `TreeMap` extrahiert werden.

Dieser Ansatz funktioniert jedoch nur für relativ kurze Messungen bzw. tiefe Samplingfrequenzen, da ansonsten der Arbeitsspeicher nicht ausreichend ist (siehe oben).

Direkt von der Festplatte

Offensichtlich müssen die benötigten Strommesswerte also vorzu von der Festplatte gelesen werden. Für einen gezielten Zugriff auf Strommessungen eines bestimmten Targets ist das vorhandene CSV-Format jedoch ungeeignet. Die darin vorkommenden Einträge (Zeit, Target, Messwert) sind weder anhand des Targets noch zeitlich geordnet. Um die Daten jedoch in einer Weise auf der Festplatte vorliegen zu haben, welche den effizienten Zugriff darauf erlaubt, wurde folgender Ansatz gewählt.

Zuerst wird die CSV-Datei (*powerprofiling.csv*) in mehrere kleinere Dateien mit einem einfachen binären Format aufgeteilt. Dabei wird für jedes vorkommende Target eine Datei erstellt, deren Inhalt aus einer Folge von Einträgen a 16 Bytes (siehe Abb. 15) besteht. Die ersten 8 Bytes entsprechen dem Zeitstempel (64-bit-Integer), die zweiten 8 Bytes dem gemessenen Stromwert (64-bit-Float). Dabei werden die Einträge entsprechend der zeitlichen Reihenfolge geordnet abgespeichert.

Werden nun für die Visualisierung die Strommesswerte eines bestimmten Targets benötigt, kann direkt auf die entsprechende Datei zugegriffen werden. Darin kann aufgrund der identischen Länge aller Einträge und deren zeitlicher Ordnung schnell navigiert werden.

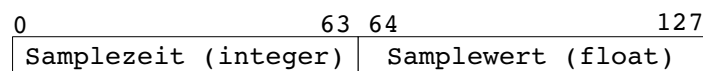


Abbildung 15: Einzelner Eintrag einer Strommessdatendatei

Das Programm, welches aus einer gegebenen CSV-Datei die entsprechenden kleineren Dateien in diesem Format generiert, wurde auch in Java, jedoch als eigenständiges Kommandozeilenprogramm implementiert. Dies im Hinblick darauf, diese Vorverarbeitung möglicherweise schon auf dem FlockLab-Server durchzuführen. Das Programm befindet sich im Verzeichnis „*PowerConverter*“ des zu dieser Arbeit gehörenden SVN-Repositorys.

Die Dateien mit den Strommesswerten im binären Format müssen in einem Unterordner „*powerprofiling*“ des FlockLab-Testlaufdatenverzeichnisses gespeichert werden, um von *Flooja* erkannt zu werden.

Beim Start von *Flooja* werden die Strommessdaten nun nicht in den Arbeitsspeicher geladen. Nur die Pfade zu den

einzelnen Dateien werden für späteren Zugriff bereitgehalten. Implementiert wird der Zugriff auf diese Dateien in der Klasse `CurrentTraceFromFile`, welche das Interface `CurrentTrace` implementiert.

Werden nun die Messwerte in einem bestimmten Zeitintervall $[t_{\text{start}}, t_{\text{end}}]$ benötigt, so wird zuerst mittels einer binären Suche in der Datei nach dem letzten Messwert gesucht, welcher vor oder direkt bei t_{start} liegt. Von hier aus wird dann die Datei Sample für Sample bis zum Ende des benötigten Zeitintervalls durchlaufen, wobei Messzeit und -wert jeweils in einem Buffer zwischengespeichert werden. Dieser Buffer kann dann an die graphische Komponente übergeben werden, um die Werte im Plot darzustellen.

Bei diesem Vorgehen ergibt sich jedoch ein Problem, dann nämlich, wenn im Plot weit herausgezoomt wird. Da das angezeigte Intervall in diesem Fall sehr lange werden kann, werden dementsprechend auch viele Messpunkte aus der Datei geladen und anschliessend geplottet, was zu spürbaren Rucklern in der Darstellung führt. Um dieses Problem zu reduzieren wurde das folgende Vorgehen implementiert.

Während der Umwandlung von *powerprofiling.csv* in mehrere einzelne Dateien wird zusätzlich für jedes Target die grösste vorkommende zeitliche Distanz zwischen zwei aufeinanderfolgenden Samples `maxSampleInterval` ermittelt. Dieser Wert wird am Ende der resultierenden binären Datei angefügt. Beim Erstellen einer `CurrentTraceFromFile` für die Strommessung eines bestimmten Targets wird dieser Wert ausgelesen und wie folgt verwendet.

Sind die Messwerte innerhalb eines bestimmten Zeitintervalls anzuzeigen, so werden nicht mehr alle Samples in diesem Intervall aus der Datei geladen, sondern einige übersprungen. Pro Bildschirmpixel genügt ein Messwert. Bei gegebener zeitlicher Zoomstufe (Zeit/Pixel) genügt es also, Messwerte mit einem bestimmten zeitlichen Abstand `maxDeltaT` zu verwenden. Die Samples in der binären Datei werden nun in Sprüngen der Grösse `floor(maxDeltaT / maxSampleTime)` durchlaufen, wobei `floor` die Abrundungsfunktion ist.

Mithilfe dieser Methode wird die Reaktionsgeschwindigkeit von Floopa vor allem bei weitem Herauszoomen deutlich verbessert.

4.2.3 Serial output

Wie oben beschrieben, enthält die Datei *serial.csv* Informationen darüber, bei welchem Observer-Target-Paar zu welcher Zeit in welche Richtung welche Textnachricht übermittelt wurde. Im Gegensatz zu Strommessung und GPIO-Tracing bzw. -Actuation sollten diese Daten in einer einzigen tabellenartigen Komponente zeitlich geordnet dargestellt werden, weshalb beim Einlesen eine einzige Datenstruktur für alle Einträge verwendet wird. Ein Eintrag wird dabei in Floopa in einem Objekt der Klasse `SerialEvent` (siehe Abb. 16) zusammengefasst. Alle Einträge zusammen werden schliesslich in einem Array zeitlich geordnet für die weitere Verwendung bereitgehalten.

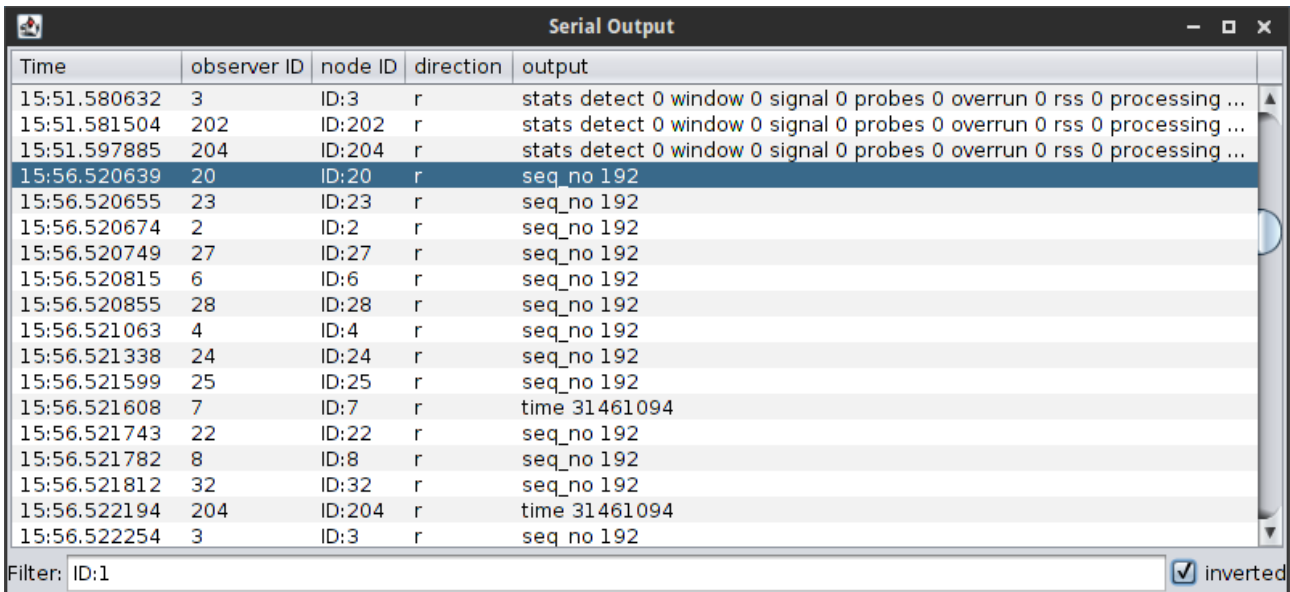
SerialEvent
<code>time: long</code>
<code>observerId: int</code>
<code>targetId: int</code>
<code>direction: bool</code>
<code>message: String</code>

Abbildung 16:
SerialEvent Datencontainer

4.3 GUI

Die Visualisierung der gewünschten FlockLab-Testlaufdaten erfolgt in Floopa aufgeteilt auf zwei Fenster: „Serial Output“ für die Daten aus *serial.csv* und „FlockLab Timeline“ für die Daten von GPIO-Tracing, -Actuation und Strommessung.

4.3.1 Serial Output



Time	observer ID	node ID	direction	output
15:51.580632	3	ID:3	r	stats detect 0 window 0 signal 0 probes 0 overrun 0 rss 0 processing ...
15:51.581504	202	ID:202	r	stats detect 0 window 0 signal 0 probes 0 overrun 0 rss 0 processing ...
15:51.597885	204	ID:204	r	stats detect 0 window 0 signal 0 probes 0 overrun 0 rss 0 processing ...
15:56.520639	20	ID:20	r	seq_no 192
15:56.520655	23	ID:23	r	seq_no 192
15:56.520674	2	ID:2	r	seq_no 192
15:56.520749	27	ID:27	r	seq_no 192
15:56.520815	6	ID:6	r	seq_no 192
15:56.520855	28	ID:28	r	seq_no 192
15:56.521063	4	ID:4	r	seq_no 192
15:56.521338	24	ID:24	r	seq_no 192
15:56.521599	25	ID:25	r	seq_no 192
15:56.521608	7	ID:7	r	time 31461094
15:56.521743	22	ID:22	r	seq_no 192
15:56.521782	8	ID:8	r	seq_no 192
15:56.521812	32	ID:32	r	seq_no 192
15:56.522194	204	ID:204	r	time 31461094
15:56.522254	3	ID:3	r	seq_no 192

Filter: ID:1 inverted

Abbildung 17: Serial Output

Funktionalität

Abb. 17 zeigt das Fenster von FlookLab, welches zur Darstellung der Nachrichten aus *serial.csv* verwendet wird. Alle Nachrichten werden in chronologischer Reihenfolge in einer scrollbaren Tabelle dargestellt. Durch Auswählen einer Nachricht (=Zeile) mit der Maus und anschliessendem drücken der Leertaste wird der Zeitzeiger des „FlockLab Timeline“-Fensters (siehe weiter unten) zur entsprechenden Zeit bewegt.

Wie bei dem bereits in Cooja enthaltenen Plugin „Mote output“ kann auch hier die Tabelle mithilfe eines regulären Ausdrucks gefiltert werden. Der durch den Anwender eingegebene Filterausdruck wird dabei auf die beiden Spalten „node ID“ und „output“ angewendet. Trifft er für eine bestimmte Zeile auf eine der Spalten zu, so wird diese Zeile bei deaktivierter bzw. aktivierter „inverted“-Checkbox angezeigt bzw. nicht angezeigt. In Abb. 17 werden durch den Filter also alle Zeilen, für welche „node ID“ oder „output“ den Ausdruck „ID:1“ enthalten, nicht angezeigt.

Da das Filtern nach Target ein typischer Anwendungsfall ist und um die Bedienung dieses Filters ähnlich derjenigen des bereits vorhandenen Plugins „Mote output“ zu halten, wird auch hier nur ein einziger Filterausdruck auf mehrere Spalten angewendet, anstatt für jede Spalte einen eigenen zu verwenden.

Im Folgenden soll auf die Implementierung dieses Fensters weiter eingegangen werden.

Implementierung

Zur Darstellung der tabellarischen Daten wird die Klasse `JTable` von Swing verwendet. Entsprechend der Architektur von Swing, bei der alle graphischen Komponenten in *Model* und *View* [3] unterteilt werden, wird auch hier die darstellende Komponente (`JTable`) mit einer Komponente verbunden, welche die darzustellenden Daten bereitstellt (`TableModel`). Über die Methode `getValueAt(int rowIndex, int columnIndex)` greift die darstellende Komponente (`JTable`) auf die darzustellenden Daten zu. Das Interface `TableModel` wird deshalb für FlookLab in einer Klasse `SerialTable` implementiert. Bei Anfrage der Daten in Spalte *j* und Zeile *i* werden die entsprechenden Daten des *i*-ten `SerialEvents` (aus dem Array aller `SerialEvents`) zurückgegeben.

Die Filterung der Einträge anhand eines regulären Ausdrucks wird durch Swing standardmässig durch die Klasse `RowFilter` unterstützt.

Bei der Implementierung dieser Komponente traten keine grösseren Probleme auf.

4.3.2 FlockLab Timeline

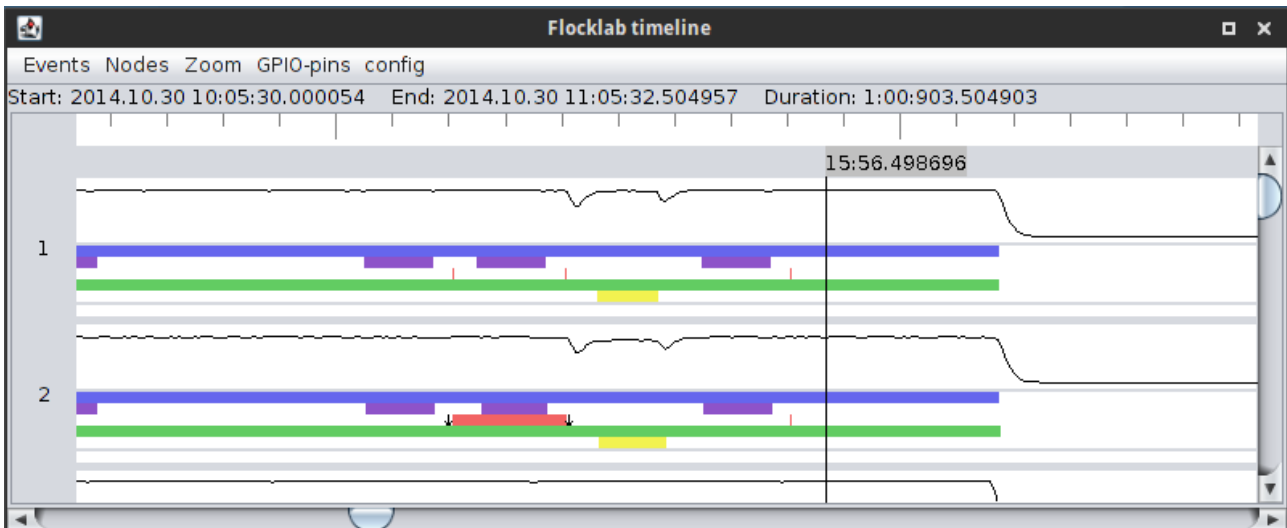


Abbildung 18: FlockLab Timeline

Im Fenster „FlockLab Timeline“ (Abb. 18) werden die Daten von GPIO-Pin-Tracing, -Actuation und Strommessung dargestellt. Anhand von Abb. 19 soll der Aufbau dieses Fensters weiter erläutert werden.

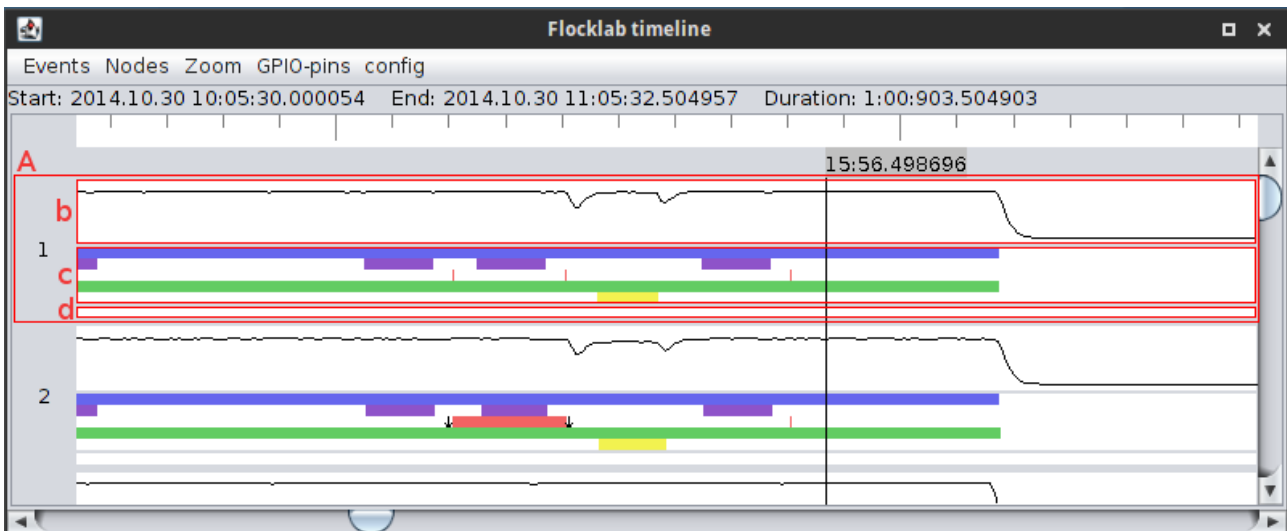


Abbildung 19: FlockLab Timeline: Teilbereiche

Für jedes Target werden dessen GPIO- und Strommessungen in einem rechteckigen Bereich (A) dargestellt. Rechts neben der Target-ID werden v.o.n.u. die Strommessung, die GPIO-Traces und die GPIO-Actuations dargestellt; bei (A) also die Strommessung (b), fünf GPIO-Traces (c) und eine GPIO-Actuation (d) für das Target mit der ID '1'. Über die untere Scrollbar kann der angezeigte Zeitabschnitt verändert werden; über die rechte kann unter den verschiedenen Targets gescrollt werden. Die Zeiteinheiten sind am oberen Rand in Schritten von 1 ms (kleine Striche) bzw. 10 ms (grössere Striche) angezeigt.

Zeitzeiger

Der Zeitzeiger (vertikale schwarze Linie) kann durch Mausklick auf eine beliebige Zeit gesetzt werden. Diese, gemessen seit Beginn des Testlaufs, wird am oberen Ende des Zeigers angezeigt. Im Fenster „Serial Output“ wird dadurch zusätzlich, falls vorhanden, die zeitlich nächstfolgende Nachricht markiert.

Beim Ziehen der Maus werden zwei Zeitzeiger sowie zusätzliche Informationen zum ausgewählten Zeitabschnitt angezeigt (Abb. 20). Dazu gehören Start- und Endzeit des markierten Zeitabschnitts, sowie dessen Dauer. Wird diese Aktion auf einem Strommessplot gestartet, wird zusätzlich der Mittelwert der Messungen in diesem Intervall berechnet und angegeben. Dafür wird über die entsprechenden Samples in der zugehörigen Datei iteriert, die Messwerte aufsummiert und schliesslich durch die Anzahl an Samples dividiert.

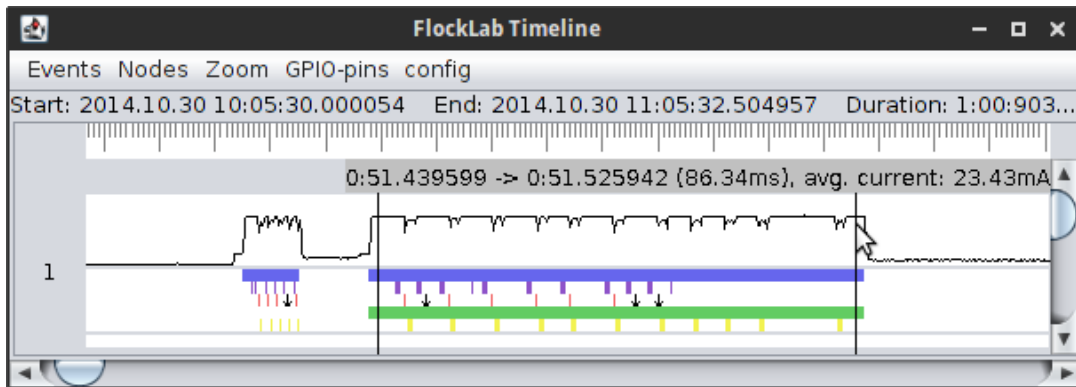


Abbildung 20: Zeitabschnittauswahl mit Strommittelwert

TimePlot

Für die Implementierung des Plots der Strommessung (b) und der einzelnen GPIO-Trace-Linien in (c) sowie der GPIO-Actuation-Linie in (d) wird in Floopja zunächst eine Klasse `TimePlot` definiert. Diese erweitert die Java-Swing-Klasse `JComponent`, welche Basisklasse jeder rechteckigen graphischen Komponente von Swing ist. Die Methode `JComponent.paintComponent`, welche für das Zeichnen der Komponente auf den Bildschirm aufgerufen wird, wird in `TimePlot` überschrieben und zeichnet nur den weissen Hintergrund. Die weiteren Operationen zum Zeichnen der Komponente werden erst in den wiederum von `TimePlot` abgeleiteten Klassen `CurrentPlot` bzw. `GpioPlot` implementiert. Jedem `TimePlot`-Objekt wird beim Erstellen ein `ObservableValue<Long>`-Objekt übergeben, durch welches der jeweilige Plot über Änderungen am angezeigten Zeitabschnitt informiert wird. Dieser Wert ist für alle `TimePlots` derselbe und wird durch die Scrollbar am unteren Rand kontrolliert.

CurrentPlot

Der Plot der Strommesswerte eines einzelnen Targets ist durch die Klasse `CurrentPlot` implementiert. Bei der Erstellung eines Objekts wird eine `CurrentTrace` (siehe oben) übergeben, durch welche auf die darzustellenden Messwerte zugegriffen werden kann. Diese werden schliesslich Linie für Linie gezeichnet.

GpioPlot

Für die Darstellung der Ereignisse einer einzelnen GPIO-Pin-Trace bzw. -Actuation wird die Klasse `GpioPlot` verwendet. Bei der Erstellung eines `GpioPlot`-Objektes wird ein `TreeSet<Long, Boolean>` ("GPIO-Trace", siehe oben) mit den entsprechenden Ereignissen übergeben. Daraus werden diejenigen Ereignisse gelesen, welche im aktuell angezeigten Zeitabschnitt liegen. Die entsprechenden Phasen, in welchen der Pin „high“ war, werden dann mithilfe der `drawRect`-Funktion als farbige Rechtecke dargestellt (siehe Abb. 19). Da in den Daten von GPIO-Tracing und -Actuation allerdings nicht auf jede positive Flanke eine negative und umgekehrt folgt, wird für diejenigen Ereignisse, welche durch die oben beschriebene Darstellung verloren gehen würden, zusätzlich ein schwarzer Pfeil nach oben bzw. unten dargestellt. Dadurch gehen bei der Darstellung keine Informationen verloren, welche in den ursprünglichen Daten vorhanden sind.

Konfiguration von "FlockLab Timeline"

Über das Menü "Events" können bestimmte Messwerte (z.B. alle Strommessungen oder alle GPIO-Traces des Pins "INT1") für alle Targets ein- bzw. ausgeblendet werden. Durch das Menü "Nodes" hingegen können alle Messungen gewisser Targets ein- bzw. ausgeblendet werden. Dieses Ein- und Ausblenden ist dadurch implementiert, dass für die entsprechende graphische Komponente deren bereits in Swing vorhandene Methode `setVisibility` mit dem richtigen Argument aufgerufen wird.

Im Menü "Zoom" kann zum einen die Zeitskala (Zooming) und zum anderen die Skala der Strommesswerte verändert werden. Dabei wird den entsprechenden `TimePlots` (`CurrentPlot` und `GpioPlot`) der neue Wert "Zeit pro Pixel" bzw. "Höchster angezeigter Stromwert" mitgeteilt, worauf diese ihre Darstellung anpassen.

Über das Menü "config" können diese Einstellungen in einer Datei abgespeichert und nach einem Neustart von Floopja wiederhergestellt werden, so dass es möglich wird, Daten von mehreren Testläufen, unkompliziert auf gleiche Art und Weise darzustellen und zu inspizieren.

Zusätzlich zu den Menüs können durch Ziehen der Target-ID-Texte nach oben oder unten die Targets umgeordnet werden.

5 Zusammenfassung und Ausblick

Mithilfe von Flooja ist es möglich die während eines FlockLab-Testlaufs erzeugten Daten, anzuzeigen und vertieft zu inspizieren. Dies gilt insbesondere auch für die grossen Mengen an Daten der Strommessungen, welche dafür jedoch zuerst vorverarbeitet und in einem Format hinterlegt werden, welches schnelleren Zugriff erlaubt.

Die graphische Oberfläche von Flooja bietet dem Benutzer mehrere Mittel, den angezeigten Ausschnitt der Daten auf den ihn interessierenden Teilbereich zu beschränken. So können zum einen die zeitliche Zoomstufe und die Reihenfolge der Targets verändert werden. Zum anderen können ganze Teilmessungen ausgeblendet (z.B. bestimmte GPIO-Pins) bzw. gefiltert (serielle Nachrichten) werden. Durch die Möglichkeit diese Einstellungen abzuspeichern und später wieder zu laden, können sie einfach für mehrere FlockLab-Testlaufdatensätze angewandt werden.

Zur Zeit muss der Benutzer die FlockLab-Resultate noch selbst über das FlockLab-Webinterface herunterladen und die Strommessdaten konvertieren. In weiteren Arbeiten könnte die Benutzerfreundlichkeit weiter erhöht werden, indem diese Schritte bereits auf dem FlockLab-Server bzw. direkt in Flooja durchgeführt werden.

6 Quellen

- [1] Talzi, I.; Hasler, A.; Gruber, S. & Tschudin, C.
PermaSense: investigating permafrost with a WSN in the Swiss Alps
Proceedings of the 4th workshop on Embedded networked sensors, 2007, 8-12
- [2] Lim, R.; Ferrari, F.; Zimmerling, M.; Walser, C.; Sommer, P. & Beutel, J.
FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems
Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN), ACM/IEEE, 2013, 153-165
- [3] A Swing Architecture Overview, <http://www.oracle.com/technetwork/java/architecture-142923.html>

SEMESTERARBEIT

für
Andreas Büchel

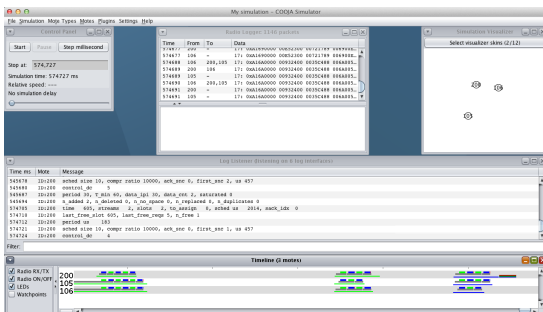
Betreuer: Roman Lim
Stellvertreter: Marco Zimmerling

Ausgabe: 22. September 2014
Abgabe: 28. November 2014

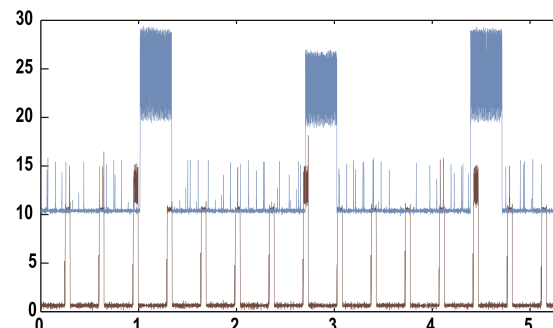
Making Sense of FlockLab Testbed Data

Einleitung

FlockLab [1] ist ein Testbed für drahtlose Sensornetze. Darunter versteht man eine Installation, die es erlaubt, Programme verteilt auf mehreren realen Sensorknoten zu testen. Typische Dienstleistungen eines Testbeds sind das Programmieren der Knoten, ein Kommunikationskanal (serielle Schnittstelle) und das Messen des Stromverbrauchs. Zusätzlich kann mit FlockLab der Zustand von GPIO-Pins erfasst und auch gesetzt werden (GPIO Tracing/Actuation). Alle diese Dienstleistungen generieren grosse Datenmengen während eines Testlaufs. Die Daten werden dem Benutzer über eine Web-Schnittstelle in Form einer Textdatei zur Verfügung gestellt.



(a) Die Visualisierung einer Simulation in Cooja. Das *Timelapse*-Plugin am unteren Rand zeigt die versendeten und empfangenen Radiopakete im Netzwerk an.



(b) Leistungsverlauf von zwei Sensorknoten.

Ziel der Arbeit

In dieser Semesterarbeit soll ein Plugin für den Cooja-Simulator [2] entwickelt werden. Mit Hilfe dieses Plugins sollen die während eines Testlaufs auf FlockLab generierten Daten komfortabel inspiziert werden können. Als Basis dienen die vorhandenen *Log Listener* und *Timeline* Plugins [3] (Abb. 1(a)). Zusätzlich sollen auch Strommessungen wie in Abb. 1(b) zu sehen angezeigt werden können.

Aufgabenstellung

1. Erstellen Sie einen Projektplan und legen Sie Meilensteine sowohl zeitlich wie auch thematisch fest. Insbesondere soll genügend Zeit für die Schlusspräsentation und den Bericht eingeplant werden.
2. Machen Sie sich mit den relevanten Arbeiten im Bereich Visualisierung von grossen Datenmengen vertraut (z. B. das Cooja-Twist Plugin oder Vizzly [4]). Führen Sie eine Literaturrecherche durch und suchen Sie gezielt nach relevanten Publikationen. Prüfen Sie welche Ideen/Konzepte Sie aus diesen Lösungen verwenden können.
3. Verschaffen Sie sich einen Überblick über das Datenformat von FlockLab und dem Aufbau von Cooja.
4. Erstellen Sie ein Konzept für das Plugin. Achten Sie dabei auf eine einfache Bedienungsoberfläche und eine effiziente interne Datenstruktur. Holen Sie zu diesem Zweck auch Ansichten von möglichen zukünftigen Benutzern des Plugins ein.
5. Implementieren und testen Sie das neu erstellte Plugin.
6. Ziehen Sie mögliche weiterreichende Funktionalität für das Plugin in Betracht. Mögliche Ideen wären 1) automatische Analysen von Traces, 2) eine Suchfunktion für Trace-Muster oder 3) eine Bibliothek für das Aufzeichnen und Wiedergeben von versendeten Radiopaketen.
7. Dokumentieren Sie Ihre Arbeit sorgfältig mit einem Vortrag sowie mit einem Schlussbericht.

Durchführung der Semesterarbeit

Allgemeines

- Der Verlauf der Semesterarbeit soll laufend anhand des Projektplanes und der Meilensteine evaluiert werden. Unvorhergesehene Probleme beim eingeschlagenen Lösungsweg können Änderungen am Projektplan erforderlich machen. Diese sollen dokumentiert werden.
- Sie verfügen über einen PC mit Linux/Windows für Softwareentwicklung und Test. Für die Einhaltung der geltenden Sicherheitsrichtlinien der ETH Zürich sind Sie selbst verantwortlich. Falls damit Probleme auftauchen wenden Sie sich an Ihren Betreuer.
- Verwenden Sie für das Projekt das bereitgestellte FlockLab SVN-Repository für alle relevanten Daten (Code, Dokumentation, etc ..).
- Stellen Sie Ihr Projekt zu Beginn der Semesterarbeit in einem Kurzvortrag vor (maximal 5 Minuten) und präsentieren Sie die erarbeiteten Resultate am Schluss im Rahmen des Institutskolloquiums (maximal 15 Minuten).
- Besprechen Sie Ihr Vorgehen regelmässig mit Ihren Betreuern. Verfassen Sie dazu auch einen kurzen wöchentlichen Statusbericht (E-Mail).
- Weiterführende Angaben finden Sie in [5].

Abgabe

- Geben Sie den Bericht in elektronischer Form (PDF) sowie alle relevanten Source-, Object und Konfigurationsfiles bis spätestens am 28. November 2014 dem betreuenden Assistenten ab. Diese Aufgabenstellung soll im Bericht eingefügt werden, genauso wie das unterschriebene Unterschriftenblatt *Plagiat* des Rektorats. Die entsprechenden Richtlinien des Rektorats sind einzuhalten.
- Die Arbeit wird anhand der Kriterien in [6] benotet.
- Die relevanten Source- und Objectfiles, Konfigurationsfiles, benötigte Verzeichnisstrukturen usw. sollen komplett im SVN-Repository abgelegt sein. Der Programmcode sowie die Dateistruktur soll ausreichend dokumentiert sein. Eine spätere Anschlussarbeit soll auf dem hinterlassenen Stand aufbauen können.

Literatur

- [1] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proceedings of the 12th international Conference on Information processing in sensor networks (IPSN)*, 2013.
- [2] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings of the 31st Conference on Local Computer Networks (LCN)*, 2006.
- [3] F. Österlind, J. Eriksson, and A. Dunkels, "Cooja timeline: A power visualizer for sensor network simulation," in *Proceedings of the 8th Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.
- [4] M. Keller, J. Beutel, O. Saukh, and L. Thiele, "Visualizing large sensor network data sets in space and time with vizzly," in *Proceedings of the 7th international Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, 2012.
- [5] TIK, "Studien- und Masterarbeiten: Merkblatt für Studenten und Betreuer." Computer Engineering and Networks Lab, ETH Zürich, Switzerland, Apr. 2009.
- [6] TIK, "Notengebung bei Studien- und Diplomarbeiten." Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 1998.