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.

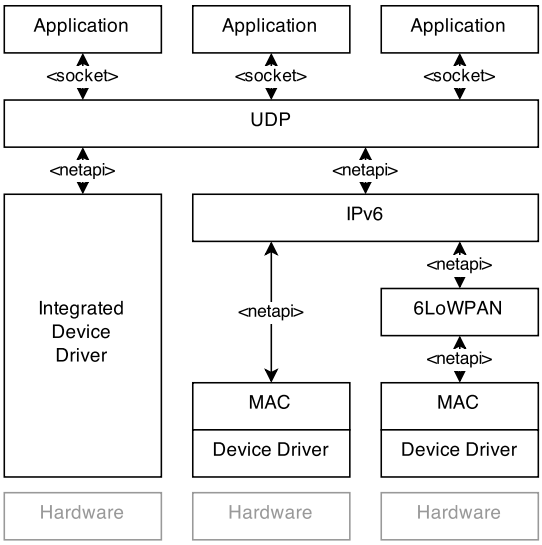

Abbildung 1. Darstellung einer beispielhaften Konfiguration des neuen RIOT Netzwerk-Stacks mit drei Netzwerkschnittstellen

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.

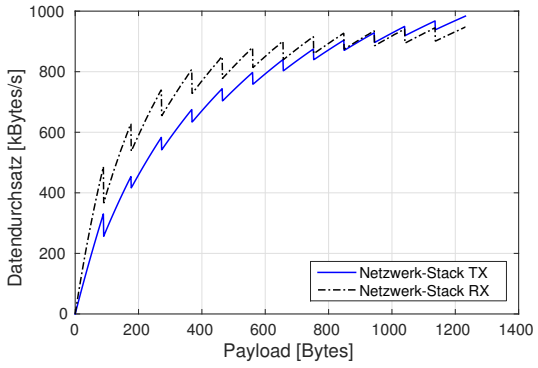

Abbildung 2. Datendurchsatz des RIOT Netzwerk-Stacks unter Ausgrenzung der Funk-Hardware

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.

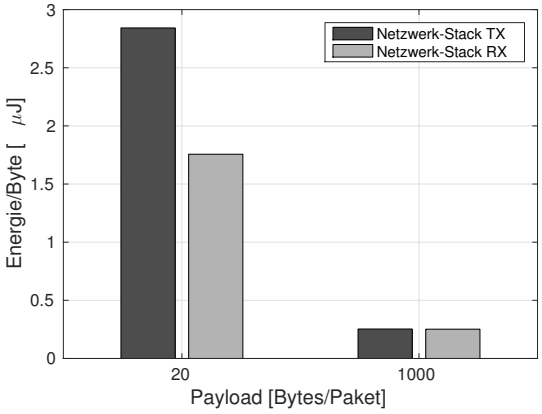

Abbildung 3. Vergleich der Energieaufnahme pro Payload-Byte im RIOT Netzwerk-Stack unter Ausgrenzung der Funk-Hardware; UART-Modul und Funk-Hardware im Wartezustand

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.

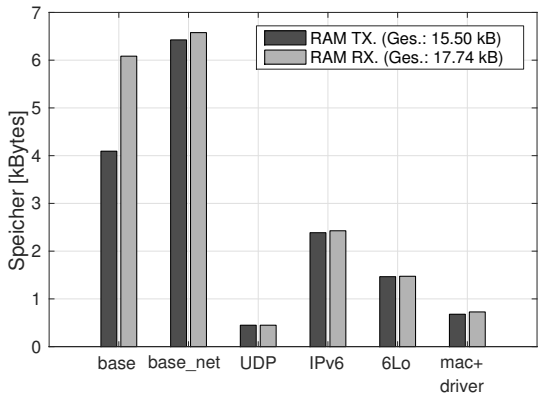

Abbildung 4. Speicherverbrauch des RIOT Netzwerk-Stacks im RAM

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.