Freie Universität Berlin

Department of Mathematics and Computer Science
Institute of Computer Science

Doctoral Dissertation

# On Information-centric Resiliency and System-level Security in Constrained, Wireless Communication

Dissertation zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.) am
Fachbereich Mathematik und Informatik der Freien Universität Berlin

vorgelegt von

**M.Eng. Peter Kietzmann**
**peter@kietzmann.cc**
**Berlin 2023**

**Erstgutachter**
Prof. Dr. Matthias Wählisch
*Technische Universität Dresden und Freie Universität Berlin*

**Zweitgutachter**
Prof. Dr. Thomas C. Schmidt
*Hamburg University of Applied Sciences*

**Drittgutachter**
Dr. Nils Wisiol
*Advanced Micro Devices (AMD)*

**Tag der Disputation**
03.06.2024

**Abstract**

The Internet of Things (IoT) interconnects many heterogeneous embedded devices either locally between each other, or globally with the Internet. These *things* are resource-constrained, *e.g.,* powered by battery, and typically communicate via low-power and lossy wireless links. Communication needs to be secured and relies on crypto-operations that are often resource-intensive and in conflict with the device constraints. These challenging operational conditions on the cheapest hardware possible, the unreliable wireless transmission, and the need for protection against common threats of the inter-network, impose severe challenges to IoT networks. In this thesis, we advance the current state of the art in two dimensions.

Part I assesses Information-centric networking (ICN) for the IoT, a network paradigm that promises enhanced reliability for data retrieval in constrained edge networks. ICN lacks a lower layer definition, which, however, is the key to enable device sleep cycles and exclusive wireless media access. This part of the thesis designs and evaluates an effective media access strategy for ICN to reduce the energy consumption and wireless interference on constrained IoT nodes.

Part II examines the performance of hardware and software crypto-operations, executed on off-the-shelf IoT platforms. A novel system design enables the accessibility and auto-configuration of crypto-hardware through an operating system. One main focus is the generation of random numbers in the IoT. This part of the thesis further designs and evaluates Physical Unclonable Functions (PUFs) to provide novel randomness sources that generate highly unpredictable secrets, on low-cost devices that lack hardware-based security features.

This thesis takes a practical view on the constrained IoT and is accompanied by real-world implementations and measurements. We contribute open source software, automation tools, a simulator, and reproducible measurement results from real IoT deployments using off-the-shelf hardware. The large-scale experiments in an open access testbed provide a direct starting point for future research.

**Zusammenfassung**

Das Internet der Dinge (IoT) verbindet eine Vielzahl heterogener eingebetteter Geräte entweder lokal untereinander, oder mit dem globalen Internet. Diese *Dinge* sind ressourcenbeschränkt, beispielsweise von einer Batterie betrieben, und kommunizieren typischerweise über verlustbehaftete Drahtlosverbindungen mit geringem Leistungsverbrauch. Kommunikation muss abgesichert werden und stützt sich dabei auf Krypto-Operationen, die oft ressourcenintensiv sind und mit den begrenzten Ressourcen der IoT-Geräte in Konflikt stehen. Diese anspruchsvollen Betriebsbedingungen auf möglichst kostengünstiger Hardware, eine unzuverlässige Drahtlosübertragung und die nötige Absicherung gegen Bedrohungen des Inter-Netzwerks stellen die IoT-Netze vor große Herausforderungen. In dieser Arbeit wird der aktuelle Stand der Technik auf zwei Ebenen optimiert.

Teil I dieses Manuskripts evaluiert den Einsatz von Information-centric Networking (ICN) im IoT. Dieses Netzwerkparadigma verspricht eine verbesserte Zuverlässigkeit bei der Datenübermittlung in ressourcenbeschränkten Rand-Netzwerken. Informationszentrischen Netzen fehlt eine Standardisierung der unteren Kommunikationsschicht. Diese ist jedoch essenziell um Geräte-Schlafzyklen und exklusiven Zugriff auf das Funkmedium zu ermöglichen. Dieser Teil der Arbeit entwickelt und evaluiert eine effektive Medienzugriffsstrategie für ICN im IoT, um den Energieverbrauch der ressourcenbeschränkten Knoten sowie die Interferenz auf dem Funkkanal zu reduzieren.

Teil II dieses Manuskripts untersucht die Leistungsfähigkeit von Hardware- und Software-Kryptooperationen, die auf handelsüblichen IoT-Plattformen ausgeführt werden. Ein neuer Systementwurf ermöglicht die Zugänglichkeit und Autokonfiguration von Krypto-Hardware durch ein Betriebssystem. Ein Hauptaugenmerk liegt auf der Erzeugung von Zufallszahlen im IoT. In diesem Teil der Arbeit werden Physical Unclonable Functions (PUFs) entwickelt und evaluiert, um neuartige Zufallsquellen bereitzustellen, die hochgradig unvorhersehbare Zahlen auf kostengünstigen IoT-Geräten erzeugen, denen es an hardwarebasierten Sicherheitsfunktionen fehlt.

Diese Arbeit nimmt einen praktischen Blick auf das ressourcenbeschränkte Internet der Dinge ein und wird von realen Implementierungen und Messungen begleitet. Wir stellen quelloffene Software, Automatisierungswerkzeuge, einen Simulator und reproduzierbare Messergebnisse aus realen IoT-Implementierungen mit handelsüblicher Hardware zur Verfügung. Die groß angelegten Experimente in einer frei zugänglichen Testumgebung bieten einen direkten Anknüpfpunkt für zukünftige Forschung.

# Selbstständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht.

Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Berlin, den 14.07.2023

Peter Kietzmann

# Acknowledgments

The start of my journey towards this dissertation was a matter of coincidence and curiosity. Underway, I was lucky to meet many people that were willing to share their knowledge with me, supported me, and motivated me. I would not have finished this journey without these people.

This endeavor would not have been possible without my supervisors Thomas Schmidt and Matthias Wählisch. Their guidance, experience, and patience was invaluable to me. And so was the financial support which never raised concerns in terms of travelling or purchase of equipment. Working in their team has taught me many things beyond the pure technical matter: The need for a detailed analysis of problems from the very root, the importance of an excellent presentation, and the relevance of sharing and discussing results with the community, which sometimes is a very cumbersome process. Thanks for giving me a chance back then, despite my differing educational path, and helping me to grow professionally.

I would like to thank Nils Wisiol and his careful feedback on Physically Unclonable Functions which has significantly helped to improve this work. Special thanks should go to Dirk Kutscher who has shared his experience and knowledge on ICN with me. Thanks for many stimulating discussions and our cooperation.

I am grateful for my colleagues of the iNET working group who became friends – for their assistance, for motivating me, for their comradeship and all the fun times throughout those years. And thank you Bobo for spreading patience in the office and companion me on many travels! Special thanks should also go to my colleagues of the ilab working group at FU Berlin who accompanied and assisted me in many studies.

A huge part of this work has been done in practical work on IoT and ICN. Hence, I am thankful for all the work that has been done on RIOT by the community, and on CCN-lite which was originally contributed by "the people from Basel". A substantial part of my experimentation effort would not exist without these tools.

Finally, I would like to express my deepest gratitude to the best family and friends in the world, for being there for me the whole time, for their unconditional love, support, and encouragement. Not only throughout my years as a PhD student, but all the years before which led me here, and those to come!

The work in this thesis was supported in part by the German Federal Ministry for Education and Research (BMBF) within the projects *I3 – Information Centric Networking for the Industrial Internet*, *PIVOT – Privacy-Integrated design and Validation in the constrained IoT*, and the HAW Hamburg project *SmartIoT*.

# Bibliographical Notes

This dissertation is based on the following seven peer-reviewed research papers.

### Part 1

- Chapter 3 is based on (alphabetical ordering of author surnames)

  C. Gündogan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN)*, pages 159–171, New York, NY, USA, September 2018. ACM. URL `https://doi.org/10.1145/3267955.3267967`

  **Conception:** I designed this paper project together with the co-authors, and conducted most of the literature review.

  **Execution:** I led the experiment execution, focusing on the deployment and measurement methodology in a large scale testbed with hundreds of constrained nodes, for which I contributed experiment automation tools, and the data curation. I implemented a significant part of the embedded software, which includes the implementation of a novel network protocol suite, that was compared to existing standard IoT-protocols.

  **Reporting:** I was instrumental in drafting the article, proofreading it, and resolving the issues raised by the reviewers. I was also responsible for the data visualization of the paper.

- Chapter 4 is based on

  P. Kietzmann, C. Gündogan, T. C. Schmidt, O. Hahm, and M. Wählisch. The Need for a Name to MAC Address Mapping in NDN: Towards Quantifying the Resource Gain. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*, pages 36–42, New York, NY, USA, September 2017. ACM. URL `https://dl.acm.org/doi/10.1145/3125719.3125737`

  **Conception:** I am responsible for the research concept. I shaped the paper project in agreement with all co-authors, and under consideration of the literature review that I conducted myself.

**Execution:** I was responsible for the entire embedded software implementation for the measurement study, as well as the scripting for the experiment deployment, experiment automation, and the data analysis.

**Reporting:** The paper was written, submitted, and edited by myself as the responsible author, supervised by the co-authors.

- Chapter 5[‡] is based on

   P. Kietzmann, J. Alamos, D. Kutscher, T. C. Schmidt, and M. Wählisch. Long-Range ICN for the IoT: Exploring a LoRa System Design. In *Proc. of 21th IFIP Networking Conference*, pages 1–9, Piscataway, NJ, USA, June 2022. IEEE Press. URL `https://doi.org/10.23919/IFIPNetworking55013.2022.9829792`

  **Conception:** I derived and formulated the research concept of this paper, based on discussions and feedback provided by the co-authors. I conducted the literature review by myself, which acted as the basis for the initial research idea.

  **Execution:** I developed the technical concept of the simulator for the experimentation. The implementation was carried out by the co-authors, who I guided in the development of the software and execution of the simulations. The data analysis was performed by myself, incorporating feedback from my co-authors.

  **Reporting:** I am responsible for the paper visualization and led the paper writing, contributing a significant part of the text. I also submitted the article and acted as the responsible author.

- Chapter 6 is based on

  P. Kietzmann, J. Alamos, D. Kutscher, T. C. Schmidt, and M. Wählisch. Delay-Tolerant ICN and Its Application to LoRa. In *Proc. of 9th ACM Conference on Information-Centric Networking (ICN)*, pages 125–136, New York, September 2022. ACM. URL `https://doi.org/10.1145/3517212.3558081`

  **Conception:** I designed the significant part of the research goals and the solution space, which is in part based on the joint research efforts of the former Chapter 5.

  **Execution:** I designed the experimental setup which is built upon real hardware, a novel custom gateway, and an existing network emulator. I implemented a significant part of the software, conducted all experiments, and performed the data analysis. Specific parts of the lower layer protocol implementations have been carried out by the co-authors, who I supervised in order to align with the overall architecture.

**Reporting:** I led the reporting and visualization, submission, and editing of the paper, as the responsible author. The text was written mostly by myself and in part by the co-authors who also provided feedback, proofreading, and corrections.

## Part 2

- Chapter 8 is based on

  P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch. A Performance Study of Crypto-Hardware in the Low-end IoT. In *Proc. of Embedded Wireless Systems and Networks (EWSN'21)*, New York, USA, February 2021. ACM. URL `https://dl.acm.org/doi/10.5555/3451271.3451279`

  **Conception:** I am responsible for the research idea, conducted the literature review, and derived the proposed research goals together with the co-authors. Based on our discussion, I designed a significant part of the solution space by myself.

  **Execution:** I implemented the majority of test code and performed the experiment execution. Parts of the embedded driver implementations for the variety of heterogeneous test devices have been carried out by the co-authors, who I coordinated for the software development.

  **Reporting:** I wrote, submitted, and edited the paper, as the responsible author. The co-authors contributed to the data analysis and provided feedback, and proofreading, which I have incorporated for the paper submission.

- Chapter 9 is based on

  P. Kietzmann, T. C. Schmidt, and M. Wählisch. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM Comput. Surv.*, 54(6):112:1–112:38, July 2022. URL `https://dl.acm.org/doi/10.1145/3453159`

  **Conception:** I developed the research idea together with the co-authors, and was responsible for the literature review. The experimental solution space was mainly derived by myself.

  **Execution:** I contributed all implementations for the experimental study and conducted the data acquisition and data curation. The analysis of the performance measurements was done by myself, and the interpretation of the statistical test results was carried out together with the co-authors.

  **Reporting:** I led the visualization of this paper, and the text was written together with the co-authors who provided feedback, proofreading, and corrections. I submitted the article and acted as the responsible author.

- Chapter 10[‡] is based on

  © 2023 IEEE. Reprinted, with permission, from P. Kietzmann, T. C. Schmidt, and M. Wählisch. PUF for the Commons: Enhancing Embedded Security on the OS Level. *IEEE Transactions on Dependable and Secure Computing*, 2023. URL `http://doi.org/10.1109/TDSC.2023.3300368`

  **Conception:** I am responsible for the research concept, the literature review, the experimental design, and the solution space. My co-authors provided feedback and suggestions, which I utilized to structure the paper.

  **Execution:** I conducted all implementations, performed the experiments in a large scale testbed, and performed the significant part of the data analysis, in discussion with the co-authors.

  **Reporting:** The paper was written, submitted, and edited by myself as the responsible author. My co-authors provided feedback, proofreading, and corrections.

[‡] In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of FU Berlin's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

# Contents

# Chapter 1

# Introduction

The Internet of Things (IoT) is evolving and an increasing number of controllers in the field is augmented with network interfaces that connect to the global Internet. The prevalent use case forecasted for the IoT consists of billions of constrained *things* that are connected via low-power and lossy wireless links. In practice, these *things* are resource-constrained embedded devices (*i.e.,* class 0–2 [58]) with limited processing capabilities, memory, and energy – perhaps powered intermittently. Operated from a small battery for as long as possible (years), common deployments aim to reduce cost and manual maintenance efforts for the great number of foreseen IoT nodes. These tiny and cheap devices are severely challenged by the current way of connecting to the Internet and require communication technologies that bridge the scale to the global internetwork. At the same time, low-power wireless transmissions introduce significant challenges such as packet loss, which interferes with service dependability and challenges communication protocols.

IoT deployments more and more distribute computational complexity to the edge of the network in order improve the network performance (*e.g.,* latency, bandwidth utilization) and to simplify content sharing in machine-type communication scenarios. This demands for a decentralized network architecture, possibly increasing the resource consumption on edge nodes. The constraints of IoT devices, and the arising communication requirements are in conflict and challenge practical IoT deployments.

Security is essential in the IoT, similarly to conventional machines that connect to the Internet. Data confidentiality, integrity, availability, and authenticity [282] rely on crypto-operations that are resource-intensive or infeasible to process on resource-constrained devices, and in conflict with limited energy resources. Random numbers provide essential input to crypto-operations, but likewise are complex to obtain [199]. In practice, many IoT deployments consist of cheap embedded devices without security features [58], and readily threaten the IoT [214] as well as the global Internet [19]. Crypto-accelerators reduce the operational overhead to enable security on modern IoT devices, but are not available on ultra-constrained platforms that are already deployed, and conflict with device cost. If available, however, hardware accelerated cryptography is a promising solution to enhance the security of future IoT deployments and contributes to nodal lifetime, but it needs to be exposed carefully, since crypto-accelerators and hardware

random number generators are occasionally vulnerable [365]. This demands for reliable (software) alternatives. Features of crypto-hardware are largely vendor specific and not sufficiently integrated into existing (software) libraries, for portability reasons. Manufacturers support their hardware but reduce flexibility which leads to a vendor lock-in. The heterogeneity of hardware platforms, software libraries, and crypto APIs threatens the IoT, because security is tied to the usability of cryptography [133]. This demands for platform-agnostic interfaces that gain hardware support for heterogeneous devices, which enables usable security for the IoT.

## 1.1 Networking the Internet of Things

Interoperability plays a central role in networked systems and enables data exchange between heterogeneous devices, independent of the manufacturer. Communication protocols establish interoperability and need to consider the constraints of IoT networks to avoid resource depletion. The choice of a transmission technology is crucial in the IoT, where nodes operate on sparse battery resources. Low-power radio transmissions, however, may introduce significant packet loss. Energy consumption and packet loss can be mitigated by the medium access layer. The higher layer protocols of the network stack enable communication across multiple hops or sites and need to scale up to larger deployments, while taking into consideration the device constraints as well as packet loss introduced by unreliable transmissions. This section summarizes common transmission technologies and protocols in the IoT.

**Low-power Wireless Technologies.** Wireless communication enables cheap and flexible deployments of numerous sensors and actuators in the IoT, compared to wired communication. Limited energy resources on battery driven embedded devices motivated the development of various optimized wireless technologies, since radio transmissions are one of the most energy consuming tasks of an IoT device. These wireless technologies are characterized by a low bandwidth, high latency, packet loss, and a below average frame size compared to common standards such as IEEE 802.11 (WiFi). Additionally, regional regulations [104] limit the utilization of certain (Sub-GHz) frequency bands and enforce radio duty cycling.

LoRaWAN [245] is a popular low-power long-range communication system for the IoT that is suitable for single-site deployments as well as for larger networks. It consists of LoRa, a physical layer (PHY) that allows for radio communication between 2 and 14 km. A highly configurable chirp spread spectrum modulation enables robust transmissions at minimal energy consumption. This modulation introduces on-air times at the order of seconds, though, and duty cycle regulations limit the effective throughput to a view bit per second in some configurations. LoRaWAN defines higher-layer protocols for LoRa. A vertically integrated network architecture organizes media access centrally and enables IoT data publishing of constrained nodes to a server-based infrastructure.

Bluetooth Low Energy (BLE) [51] introduces networking between embedded sensors or actuators and mobile devices such as smartphones, tablets, or notebooks. BLE offers two modes of

operation: a connection-oriented point-to-point mode and a connection-less advertising mode, both targeting single-hop personal area networks (PANs) at 1 Mbps throughput. BT mesh [52] is standardized on top of BLE, enabling multi-hop topologies, many-to-one, and many-to-many group communication.

IEEE 802.15.4 [174] is a popular PAN standard for industrial systems and defines low-power and low-rate physical layers as well as media access control (MAC). The MAC layer specifies different operation modes which can be fully contention-based using Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) for media access, or utilize time- and frequency multiplexing. The IEEE 802.15.4e specification amendment introduces Time Slotted Channel Hopping (TSCH) which is used by popular automation systems such as ZigBee [417], ISA100.11a [178], or WirelessHART [390]. Another operation mode is the Deterministic and Synchronous Multichannel Extension (DSME), providing additional features to the synchronized slotframe structure in order to improve reliability and energy consumption.

**Standard Networking Protocols for the IoT.** The Internet Engineering Task Force (IETF) has designed a suite of protocols for serving the needs of constrained IoT networks. 6LoWPAN [272] is an adaptation layer that enables IPv6 communication over constrained IEEE 802.15.4 links, optionally with 6TiSCH [394, 380, 391] to employ time-slotted media access. RPL [399] defines IPv6 routing for low-power and lossy networks and arranges nodes in a multi-hop mesh topology. IPv6 over BLE [278] defines a mapping to enable IP-connectivity on a single-hop BLE link.

The Constrained Application Protocol (CoAP) [341] offers a lightweight alternative to HTTP while running over UDP, or DTLS [313] for session security. TCP is traditionally too complex for constrained nodes, but migrating to the IoT as well [215, 129]. This set of solutions extends the host-centric paradigm of the Internet to the embedded world of constrained nodes, and puts IPv6 in place for loosely joining these *things*. By assigning an IP-address to each constrained node, their resources become addressable through the Internet, on the end-to-end path. CoAP got extended by Object Security for Constrained RESTful Environments (OSCORE) [335] which slowly starts to break the host-centric Internet paradigm into a data-centric approach, securing data objects rather than end-to-end communication channels like in stream oriented protocols such as (D)TLS.

Adaptation layers for IPv6 over low-power wide area networks [107] (*e.g.,* LoRaWAN) evolve and compress large CoAP and IPv6 packets using Static Context Header Compression (SCHC) [265, 266], to cater to heavily reduced frame sizes. The underlying long-range network still inherits all its original properties, though [124].

**Alternative IoT Protocols.** While the IETF community standardizes IoT protocols, companies deploy Message Queuing Telemetry Transport (MQTT) [36], which is adopted by many cloud services [98]. The protocol implements a publish-subscribe paradigm through a centralized broker, which decouples data sinks and sources. An IoT version of that protocol, namely MQTT-SN [352], reduces memory and bandwidth requirements to save resources on constrained

IoT nodes. Similar to CoAP, MQTT-SN uses a connection-less UDP transport, instead of a notably more complex TCP connection which introduces overhead on constrained nodes.

LwM2M [350] is an application layer protocol for machine-to-machine (M2M) communication. Standardizing device management, access control, semantic data interoperability, and software update features, it contributes to device interoperability and secured resource access. Constrained devices act as clients and communicate with non-constrained servers. Cloud applications interact with clients via these servers, imposing a server-centric architecture. Different transport bindings for LwM2M exist and inherit the performance capabilities of the underlying protocols: CoAP, HTTP, or MQTT.

**Information-centric Networking.** Doubts in the research community arose whether host-to-host sessions of IP are the appropriate approach in disruption-prone and lossy wireless IoT networks, and the data-centric nature at the Internet edge called for rethinking the current IoT architecture [331]. ICN networks [7] have been identified as promising candidates to replace the rather complex IETF network stack. A hop-wise data transport and in-network caching as contributed by Named Data Networking (NDN) [180, 413] bear the potential to increase robustness of application scenarios in regimes of low reliability and reduced infrastructure. Following initial concepts [289] and early experimental work [33], the adaptation, analysis, and deployment of NDN for the IoT became an active research area [219] that advocated the IoT as a candidate of NDN adoption.

## 1.2 Securing the Internet of Things

System security depends on data confidentiality, integrity, authenticity, and availability [282] as well as the usability of the interfaces [133, 3, 267, 300], which is challenged by the heterogeneity of hardware- and software implementations in the IoT. Deployments of constrained networks more and more utilize an operating system [97, 98] to make its security benefits accessible to a wide range of IoT devices. IoT applications built on top of an operating system (OS) benefit from reduced implementation overhead and enhanced dependability as they reutilize existing, well-tested code such as network stacks, drivers, or crypto-libraries. The abstraction layer of an operating system acts as an entry point for a consistent crypto-integration on different layers, to prevent security and performance pitfalls, while possessing a potentially high developer familiarity through the interface. This section introduces hardware- and software security services of an IoT operating system.

**Implementation of Crypto-operations.** The IoT network stack utilizes cryptographic operations to protect data, resources, and identities. Cryptographic keys and nonces are essential input to these operations and need to be stored in flash memory or RAM, which challenges resource-constrained class 0 devices that barely provide enough memory to operate a network stack without security [56]. Common cryptographic algorithms are very complex to process on these devices and burden the CPU and battery. While researchers tweak crypto-software

libraries or configurable hardware implementations (*e.g.,* on FPGAs) in order to improve the performance, and to resist side channel attacks, chip vendors more and more add crypto-accelerators to their microcontrollers, which improve the performance of crypto-operations executed on resource-constrained devices. In parallel, similarly to trusted platform modules (TPMs) [136] for personal computers, so called 'secure elements' evolve in the IoT and provide external co-processors and tamper protected key storages. Keys are produced, stored, and consequently never leave these devices, increasing their privacy and integrity. Secure elements act as an isolated crypto-processor and operate on the keys derived thereon, keeping the secret material in a protected storage for a whole lifetime. This shifts the user access to keys towards an ID-based paradigm. This hardware and software diversity across platforms of multiple vendors, and the plethora of crypto-libraries poses challenges on a system integration which should (*i*) utilize crypto-hardware features optimally, in order to protect delicate key material, (*ii*) operate frugal on limited battery resources, and (*iii*) increase usability through a unified interface, which contributes to systems security.

**Generation of Random Numbers.** Random numbers are essential in computer systems to enfold versatility and to enable security. Almost every operating system provides ways to generate random numbers. Unfortunately, misconceptions about randomness are common in the design and implementation of operating systems [78]. More than a dozen open and closed source systems exist that can operate various classes and types of devices. This work is motivated from the heart of the resource-constrained IoT. Our insights about the creation of random sequences in the IoT can be applied to all types of devices, though.

True random sequences are generated from random physical processes. Collecting these values is challenging because underlying processes are slow or do not output continuously, increasing the resource consumption on constrained nodes. Furthermore, physical device access opens a side channel attack vector. Tampering with environmental conditions may vary the randomness properties of the underlying process. True random number generators (TRNGs) exist on many IoT platforms as a hardware peripheral – raising the question of the randomness quality of a forced sequence [365, 82]. In contrast, pseudo-random number generators (PRNGs) are deterministic algorithms that output sequences with random properties. Cryptographically secure PRNGs (CSPRNGs) provide uniformly distributed random numbers for security services and remain unpredictable with a security strength [39] of the underlying algorithm (commonly 128, 192, 256 bits). If seeded with sufficient entropy that meets a desired security strength, (CS)PRNGs expand the seed into long sequences of random numbers, without depending on true random samples at all times.

**Execution of Physical Unclonable Functions.** The notion of a Physical Unclonable Function (PUF) dates back to Pappu *et al.* [297], and Gassend *et al.* [121], who describe a technique to identify integrated circuits based on individual and intrinsic device properties, introduced by physical process variations during manufacture. PUFs are a promising class of solutions to provide unique identities, or non-uniform keys across similar devices. At the same

time, these physical structures are affected by noise, contributing randomness on a single device. A body of work has identified SRAM memory as a suitable PUF [173, 332, 376, 326]. The startup state of uninitialized memory forms a device unique pattern which acts as a digital fingerprint. This enables extraction of unpredictable (but reproducible) secrets, as well as random seeds, which links to random number generation as introduced previously.

SRAM PUFs are particularly attractive for the constrained IoT, since almost every off-the-shelf platform embeds SRAM on the microcontroller. Low-cost platforms that lack hardware security features (*e.g.,* a TRNG for seeding) can thus implement PUFs in software without increasing device cost. This enhances the security for a wide range of nodes that are already deployed, or will be. The application of software PUFs is not limited to low-cost platforms, though, but can also bootstrap security hardware such as trusted execution environments [255].

## 1.3 Research Questions

### 1.3.1 Robust and Energy-efficient Wireless Edge Communication

**Potentials of ICN for Constrained IoT Networks** and **MAC Address Mapping in ICN**. Low-power and lossy transmissions in wireless IoT networks challenge data reliability. More severely, error probabilities accumulate in multi-hop networks. Standard IoT protocols (*e.g.,* CoAP, MQTT-SN) address the constraints of IoT networks and apply retransmissions after a loss. These protocols base on IP-networking and inherit the underlying end-to-end transport properties. ICN operates fundamentally differently and contributes a hop-wise data transport. Hop-wise retransmissions and caching potentially reduce link load as well as cross traffic. These features are particularly attractive for low-power networks that are heavily affected by lossy wireless links. ICN lacks the concept of a link layer, though. Interests are usually broadcasted, contributing to path diversity. Blind broadcast forwarding, however, conflicts with limited device resources and excludes reliability mechanisms of common low-power radios, that are only employed on unicast (MAC) traffic. The potential benefits of a hop-wise transport in ICN, together with the surprisingly unexplored field of an ICN MAC layer mapping lead to the following research questions, which are further discussed in Chapter 3 and Chapter 4.

**Research Questions**

- Can ICN increase reliability and reduce energy demands in lossy wireless networks, compared to standard IoT protocols?
- What is the resource overhead of using broadcast in constrained wireless ICN networks, instead of unicast?

**Decentralized MAC and Network Layer for LoRa**. The LoRa PHY provides attractive properties for the IoT (*i.e.,* energy-efficient long-range communication), but the challenging wireless domain led to a network design (LoRaWAN) that is unreliable and prone to wireless in-

terference, largely uplink oriented, and highly centralized, requiring a permanent infrastructure backhaul to a network server. The potential benefits of a wireless MAC layer that enables robust communication for battery operated IoT devices, and the immanent hop-wise caching features of ICN lead to the following research questions, which are further discussed in Chapter 5.

**Research Questions**

- How can we enable reliable and bidirectional LoRa wireless transmissions?
- How can we decentralize LoRa networks in order to simplify data sharing at the edge, without requiring a permanent infrastructure backhaul?
- Can ICN serve the needs of decentralized LoRa networks and contribute to resource efficiency through the caching capabilities?

**Delay-tolerant Networking with ICN**. Edge deployments (including LoRa networks) may introduce untypically long, and varying round trip times that operate on a different timescale compared to application-agnostic forwarders or routers in a fast network, *i.e.,* the Internet. Additionally, data generation in the IoT commonly happens sporadically, which challenges request-driven protocols. In ICN networks, forwarding state might expire prematurely, which prevents forwarding and effective content caching. This quickly leads to inefficient polling, and wasteful retransmissions when round trips are untypically long, or data is not available. The benefits of ICN for slow edge networks, possibly with intermittent connectivity, and the yet unspecified Interest retransmission procedure in ICN lead to the following research questions, which are further discussed in Chapter 6.

**Research Questions**

- How can we interconnect networks of different time domains effectively, *e.g.,* wired networks, and edge networks that expose much longer delays?
- Can we improve the efficiency of LoRa networks by utilizing ICN caches?
- How can we achieve reactivity to sporadic IoT data in ICN, without application-awareness on networked nodes?

### 1.3.2 System-level Security on Constrained Embedded Devices

**Analysis and Integration of Cryptographic Backends**. Commercial off-the-shelf IoT platforms more and more include hardware security features to reduce the energy consumption of cryptographic operations and to improve key privacy. Platforms without crypto-acceleration utilize one out of many available software libraries. An operating system should provide seamless support to crypto-functionality, but the heterogeneity of supported hardware- and software features imposes challenges when integrated below an OS abstraction layer. Common IoT oper-

ating systems still lack a seamless crypto-integration with hardware support, and often rely on a single security software library. Manufacturer SDKs possibly provide an OS for their hardware but reduce flexibility towards a vendor lock-in. Existing performance analyses are algorithm specific and often evaluated bare metal, without an IoT OS. The lack of a consistent and vendor independent crypto-integration, and missing resource analyses across crypto-hardware and software libraries lead to the following research questions that will be further addressed in Chapter 8.

**Research Questions**

- How can we make crypto-hardware and the variety of crypto-software libraries uniformly accessible in an IoT OS, without exposing the backend configuration and the choice between numerous crypto APIs to the user?
- How significant is the resource advantage of crypto-hardware on commodity IoT platforms, compared to crypto-software operation?

**Random Number Generation in the Low-end IoT**. Many daily routines of an IoT device consume random numbers and impose differing requirements on their generation. Randomized sensor sampling, or reinforcement learning, require fast and efficient random number generation at little energy consumption. Cryptographic key generation must be fully unpredictable, which involves resource-intensive crypto-operations that should be executed carefully on battery driven nodes. Hardware random number generators are available on many platforms, but are (*i*) not always the most efficient solution, and (*ii*) occasionally vulnerable. Pseudo-random algorithms are a viable alternative, but they require a random seed for initialization, which relies on sampled physical processes – a challenging and highly platform specific task. Instead of leaving random number generation to the user, we argue that an operating system should provide a flexible randomness subsystem. At the same time, misconceptions about randomness are still common in the design and implementation of operating systems [78]. The following research questions result from this problem statement and are further addressed in Chapter 9.

**Research Questions**

- What are the threats to random subsystems of IoT devices?
- Which random source serves IoT needs in terms of statistical properties and resource requirements on constrained embedded devices?
- How can we integrate a random subsystem into an IoT OS, considering hardware diversity, resource constraints, and randomness requirements?

**Seed- and Key Generation with Physical Unclonable Functions**. Cryptographic systems utilize CSPRNGs, seeded with input of high entropy, and tamper-resistant unpredictable device identities. Both are hard to obtain on low-end IoT devices that lack hardware security

features. SRAM PUFs are a promising class of solutions to this problem, acting as a digital device fingerprint. SRAM is available on most platforms, which makes it an attractive solution to enhance the security of IoT devices, without adding hardware cost. Quantifying subtle statistical effects of SRAM-derived numbers requires a comparative analysis between large quantities of devices. The benefits of a zero-cost solution that enhances device security, and a yet missing statistically significant sample size for testing the unpredictability of the SRAM PUF lead to the following research questions, which are further discussed in Chapter 10

**Research Questions**

- How can we derive unpredictable numbers on low-cost devices?
- How unpredictable are the numbers derived from SRAM PUFs?
- What are the limitations of SRAM PUFs in an all-day IoT deployment?

## 1.4 Methods

The IoT is a challenging domain in terms of the evaluation of resource-constrained embedded systems. The absence of human-machine interfaces, the lack of existing monitoring tools, and the constraints of devices whose performance is directly affected by adding measurement probes complicate the assessment of performance metrics during regular device operation. On the other hand, the device constraints upfront motivate an evaluation on realistic IoT hardware in order to face memory-, processing-, and energy bottlenecks, instead of emulating these platforms on less constrained computers. The large number of networked nodes introduces a notable engineering overhead in terms of experimental deployment efforts and data acquisition. Still, this work aims to attain experimental data from realistic deployment scenarios.

Protocols that are designed to connect tens or hundreds of nodes in a multi-hop topology, partially within wireless reach, demand for an equal evaluation. Bandwidth demands, wireless interference, as well as memory and processing load increase and challenge not only data transmission, but also software implementations that are often hardly tested in large-scale scenarios [401, 42]. Wireless fluctuations and cross traffic in license free frequency bands motivate a realistic evaluation of these systems, since they can hardly be estimated theoretically. Still, emulators and simulators can gain initial insights prior to real world experiments, when a deployment incurs high deployment cost. The remainder of this section introduces our evaluation methods and the utilized hardware and software environments. By using open access testbeds, open source software, and commercial off-the-shelf hardware platforms, all evaluations are fully reproducible and facilitate seamless future research thereon.

**Experiments in a Testbed.** The experiments of Chapter 3, Chapter 4, and Chapter 10 were conducted on the FIT IoT-LAB testbed [5] to attain a large number of real IoT nodes. The testbed consists of seven sites with different topologies and a total number of more than

1500 nodes of 25 architectures. This enables flexible deployments of single-hop and multi-hop networks. *M3* boards make up the majority of nodes and reflect properties of commercial off-the-shelf class 2 devices. They consist of a 32-bit ARM Cortex-M3 CPU, integrated into the STM32F103REY microcontroller (MCU), which operates at 72 MHz and provides 64 kB of embedded SRAM and 512 Bytes internal flash. Additionally, these nodes are equipped with various sensors and an 802.15.4 transceiver, enabling network connectivity. These transceivers implement basic MAC layer functionality in hardware, namely automatic ACK handling, frame retransmission (*i.e.,* ARQ), and CSMA/CA. Each constrained node is attached to a control node which provides an energy monitor (INA220) and an additional 802.15.4 radio for wireless packet monitoring. The testbed offers additional tooling for experiment monitoring and control, and an aggregator that collects nodes serial output, acting as a central time reference. This enables time measurements across nodes when quantifying protocol timing performances.

**Local Experiments.** The experiments of Chapter 6, Chapter 8, and Chapter 9 were executed locally, using commodity IoT devices and external SoCs (*i.e.,* LoRa transceivers, secure elements), to attain as heterogeneous platforms as possible, with varying hardware-features. Local deployments allow for attaching more accurate measurement instruments, compared to the remote testbed. For measuring time, a logic analyzer samples at 12 MS/s by toggling an I/O pin via direct register access on the test device. To measure electrical current, we connect the test platform to a regulated voltage supply (Siglent SPD3303C) and evaluate the current consumption of each operation using a digital sampling multimeter (Keithley DMM7510 7 1/2) at 1 MS/s. A measurement period is marked by toggling I/O pins. We connect our probes in series with the MCU and turn off unused hardware components (by hardware switches or in software), to bypass unrelated current flows.

**Simulations and Emulations.** The contributions of Chapter 5 base on a simulative assessment of the proposed network architecture, to explore the feasibility and performance potentials prior to programming and deploying real devices. We conducted the simulations in the discrete event simulator OMNeT++ [377] for building network simulations. The INET framework [176] is an OMNeT++ model for wired, wireless and mobile networks and includes common network stacks and simulation utilities, *e.g.,* traffic generators. We found existing implementations that act as a starting point for our model, namely ccnSim [71] for ICN support, openDSME [186] which implements the 802.15.4 DSME standard, and FLoRa [349] as wireless propagation model to simulate LoRa networks.

Our implementation and the configurations applied in Chapter 6 base on the former simulative assessment. To emulate a fast network like the Internet in Chapter 6, *Mininet* [268] creates virtual networks of varying topologies and link properties, and connects our gateway, implemented on real IoT hardware, to the emulated network, via a virtual TAP (Test Access Point) bridge.

**Open Source Software.** Protocol comparisons, the assessment of hardware- and software

cryptography on heterogeneous devices, and the need for driver support of differing IoT technologies demand for an established and versatile code basis. We base all experiments (except simulations) on the IoT operating system RIOT [34]. The open source operating system supports numerous commercial off-the-shelf platforms through the hardware abstraction layer, which facilitates code reusability across heterogeneous platforms. Instead of re-implementing all functionally for this work, the existing code base acted as a staring point, and the solutions developed in this work were successively integrated into the operating system.

The GNRC networking subsystem in RIOT [230] provides a complete UDP/IPv6/6LoWPAN network stack and enables the integration of new protocols through a layered software design. In that way, many protocols were integrated already: CoAP, MQTT-SN, LwM2M, LoRaWAN, *etc.* . The inclusion of the CCN-lite [372] stack enables information-centric networking, the staring point for Part I.

In the beginning of this work, RIOT did not provide a subsystem to access cryptographic operations systematically, but many external software crypto-libraries that integrate via the package system. The hardware support for varying IoT boards and different crypto-software libraries enabled initial performance comparisons, which was the starting point for Part II.

**Statistical Test Suites.** Quantifying the properties of random numbers requires thorough statistical analyses of large test sequences. Instead of re-implementing common tests that are described in the literature [205], Chapter 9 applies three different open test suites: (*i*) NIST STS [43] which, however, has been rejected for the assessment of cryptographic random numbers in the meantime[1], (*ii*) DIEHARDER [61], and (*iii*) TestU01 [225]. They analyze the output of different random sources from various IoT platforms against the hypothesis of uniform randomness.

## 1.5 Contributions and Document Outline

Figure 1.1 outlines the organization of this thesis. Our contributions address the research questions of Section 1.3 and are organized in two main parts, which focus on wireless information-centric communication, and the security of constrained embedded devices in IoT networks.

**Part I** assesses information-centric networking for the IoT. In Chapter 3 we compare information-centric and host-centric IoT-protocols in low-power and lossy networks. We conduct a comparative evaluation at large-scale, using resource-constrained IoT nodes in a testbed. Our experiments arrange nodes in single-hop topologies to attain a baseline measurement, and in multi-hop topologies, which increase network stress, wireless interference, and loss by parallel data flows. We assess how hop-wise data replication in ICN, compared to end-to-end principles, affects the network performance and reliability. In Chapter 4 we utilize the same deployment option and focus on the media access strategy in wireless ICN networks. Our systematic resource

[1] `https://csrc.nist.gov/news/2022/decision-to-revise-nist-sp-800-22-rev-1a`

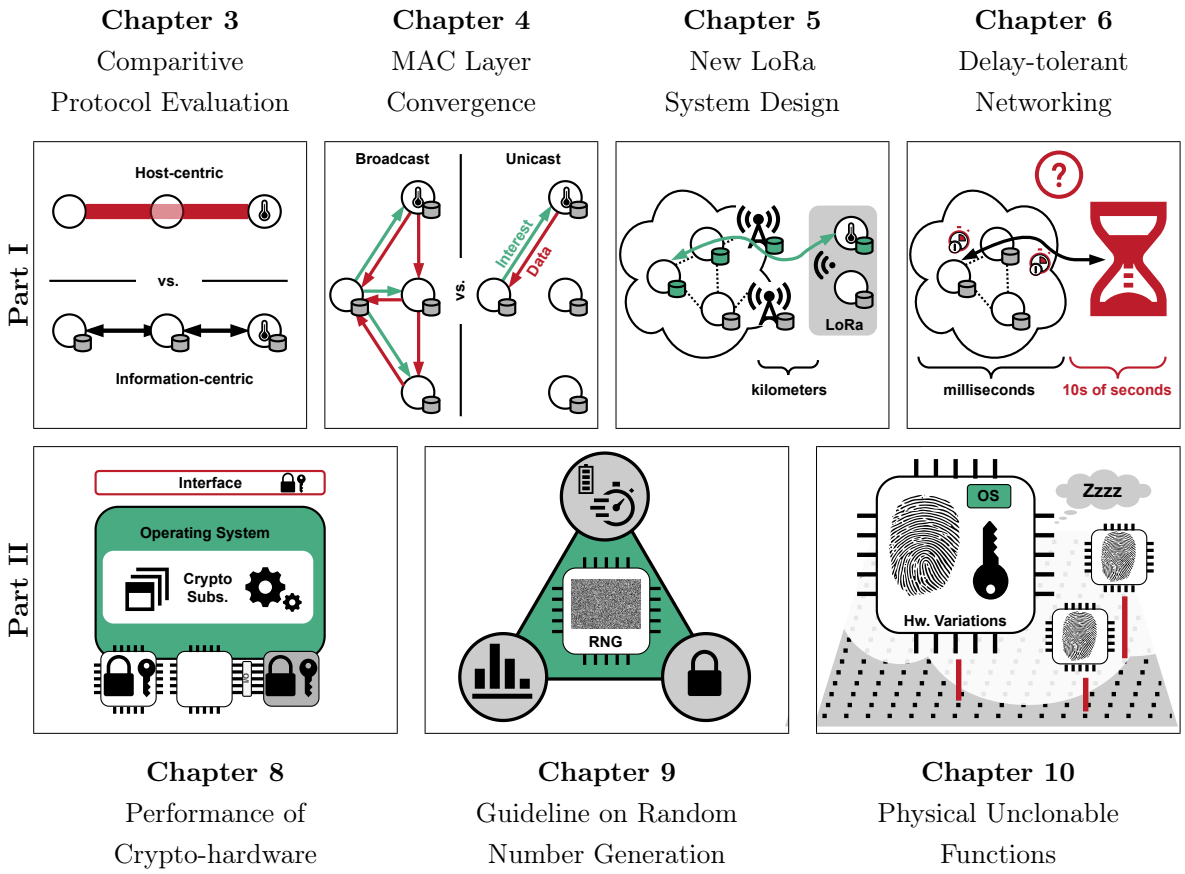



Figure 1.1: Overview of the parts and chapters included in this thesis.

analysis quantifies the impact of the name to MAC address mapping. A special focus lies on the system level overhead on resource-constrained nodes which are challenged by broadcast forwarding, increasing active CPU time, radio utilization of nodes in reach, and hence, their energy consumption.

In Chapter 5 we utilize the former experiences to design a new LoRa communication system which leverages (*i*) the DSME MAC to enable reliable bidirectional transmissions and device sleep; (*ii*) ICN to benefit from a decentralized network architecture at the edge, without the need for a permanent infrastructure backhaul. We contribute a simulation model in OMNeT++, and assess transmission options for Interest/data packets using DSME primitives. A novel gateway design serves one LoRa stub network and acts as router to connect constrained nodes to the Internet. In Chapter 6 we argue that practical application-agnostic ICN forwarders on the Internet are challenged by long and differing producer delays, *e.g.,* introduced by slow LoRa round trips. We define new gateway behavior to act as a custodial node, which includes ICN caches that allow battery driven endnodes to maximize sleep, *i.e.,* to save energy. To analyse the resulting system, we deploy our implementation of DSME-LoRa with two ICN extensions

that handle various producer delays, and conduct experiments on off-the-shelf IoT hardware, and an emulated high-speed Internet.

**Part II** assesses the OS-level accessibility and performance of cryptographic building blocks, on heterogeneous off-the-shelf platforms in the low-end IoT. In Chapter 8 we categorize the landscape of crypto-implementations in the IoT and present an integration concept to uniformly access heterogeneous software and hardware security implementations, *i.e.,* accelerators and key storages, through a user-centric interface. We contribute an implementation of the crypto-subsystem to RIOT, utilized to undertake an OS-level analysis across various IoT devices, and we quantify the advantage of crypto-hardware compared to software in terms of processing time, energy consumption, and memory footprint.

In Chapter 9 we survey taxonomies and threats to randomness subsystems in the context of the IoT. Based on statistical analyses and performance benchmarks of hardware and software (pseudo-) random number generators, we design an OS-level randomness subsystem for RIOT. Our integration concept considers differing randomness requirements, diverse hardware properties, and resource constraints of common IoT devices. This allows us to deduce general recommendations for the design of randomness systems.

In Chapter 10 we contribute an analysis of uninitialized SRAM memory on IoT nodes that aged naturally in the deployment of a testbed. Our sample size of more than 700 devices closes a research gap of missing statistically significant sample sizes for testing the unpredictability of uninitialized memory pattern. These measurements lead to a design and the integration of SRAM PUFs in RIOT, which generate two random seeds that initialize pseudo-random number generators, and a secure key per device, to bootstrap the cryptographic subsystem during startup. Our OS-level integration enables these features for a wide range of heterogeneous IoT devices via the hardware abstraction layer, and provides an additional mechanism to protect operation after a soft-reset. We analyze threats, and identify the limitations of SRAM PUFs from the perspective of an all-day IoT deployment.

# Part I

# Robust and Energy-efficient Wireless Edge Communication

# Chapter 2

# Motivation and Problem Statement

## 2.1 Protocols for Data Retrieval in the IoT

**Prominent IoT Protocols.** CoAP [341] is the IETF standard to implement data transfer in the Internet of Things. It replaces HTTP/TCP in the traditional IP-stack by a REST service interface on the application layer, and UDP as lean connectionless transport layer for resource-constrained machine-type communication. Three primitive protocol mechanisms provide flexible machine interaction: (*i*) pull implements a request-response pattern, (*ii*) push enables event-based communication, (*iii*) and observe resembles a publish-subscribe logic.

MQTT is a protocol for machine-to-machine communication. It requires an ordered lossless connection capability, typically provided through TCP/IP, which is inappropriate for the constrained IoT. MQTT-SN is a lightweight alternative to MQTT for sensor networks, which utilizes a connectionless UDP transport and reduces header sizes to fit small packet MTUs of low-power radios. Both MQTT variants implement a publish-subscribe pattern via a central broker. This enables loose coupling between sensors (typically publishers) and applications or actuators (typically subscribers). An asynchronous messaging model reflects event-based IoT data generation naturally, and the centralized broker facilitates multi-party subscriptions, possibly without increasing the communication overhead on constrained nodes. Both protocols, CoAP and MQTT, rely on IP-based host-to-host sessions and are still challenged by the underlying disruption-prone lossy regime. Doubts arose in the research community, whether the end-to-end paradigm is the appropriate approach in lossy IoT networks [331, 155].

**Information-centric Networking for the IoT.** Information-centric networking is a future Internet paradigm that supersedes the host-centric IP-based infrastructure with a data-centric approach. ICN decouples data from its location. Hence, data is accessed only via names instead of endpoints that store or produce it. Essentially, the decoupling of data from its origin, universal content caching, and an inherent content replication mechanism aim to increase scalability, reduce bandwidth demands, and facilitate content distribution in the core Internet, or edge networks.

Different ICN-protocol variants arose, one of which is Named Data Networking (NDN) [180, 413]. If not mentioned otherwise, in the remainder of this thesis, we refer to NDN when

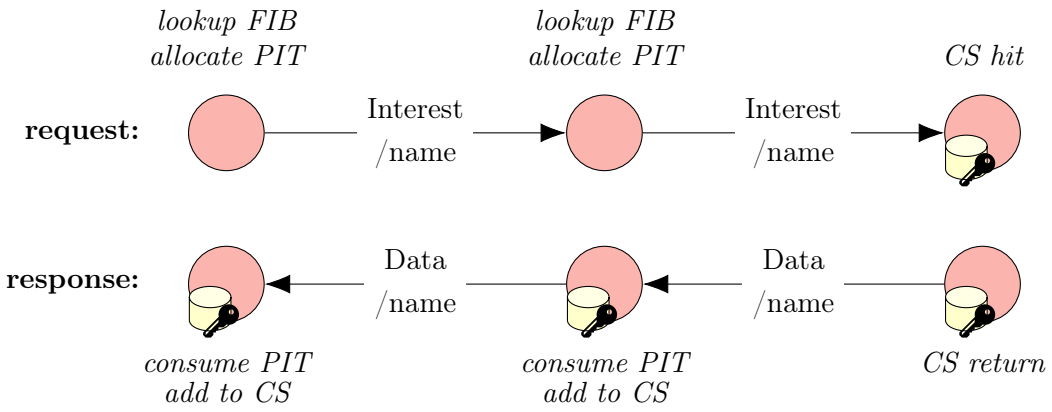
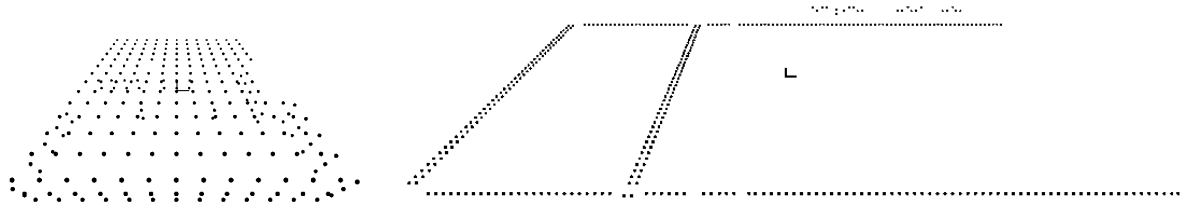



Figure 2.1: Forwarding of Interest/data packets in NDN, and common data structures to save routes, forwarding state, and content.

writing ICN. Figure 2.1 visualizes the NDN protocol logic. Two message types, namely Interest and data, implement a request-response paradigm. Interests contain the requested data name and are forwarded to one or multiple interfaces (namely *faces*) according to the Forwarding Information Base (FIB). On incoming Interest, each forwarder performs a cache lookup in its Content Store (CS) and serves the request with data, if available. Otherwise, the forwarder performs a lookup in the Pending Interest Table (PIT), which contains temporary forwarder state of previously forwarded Interests. If the data name has been forwarded before (within a defined time epoch), the request is aggregated and terminates. Otherwise, on missing CS and PIT entry, the Interest is forwarded according to the FIB, and forwarder state in the PIT is created which contains the requested content name as well as the incoming interface. On a cache hit, the data packet follows the reverse path noted in the PIT entries of each forwarder. Data consumes the PIT entry and is cached on every hop (according to caching rules) until it reaches the original requester. Consequently, a subsequent Interest for that content item can be served from any node that caches it. Unsatisfied Interests trigger retransmissions of the request after a loss. Retransmissions are repeated until the consumer application receives the requested content, or the retransmission procedure terminates. Retransmitted Interests, however, can potentially be served by a neighbored cache.

The inherent Interest aggregation, and in-network caching provide the unique potential to enable robust communication to nodes in disruptive edge networks, and to reduce link load. Data replication (*i.e.,* decentralization) creates redundancy and simplifies data sharing. Still, open problems persist, namely naming, routing, and forwarding [387, 219]. Retransmission procedures are yet unspecified. Producer initiated data transfer [219, 339] contradicts the consumer driven ICN paradigm, but it appears essential in asynchronous event-triggered and slow-acting IoT use cases.

**Protocol Comparison.** Chapter 3 provides a thorough comparative analysis of the three pro-



(a) Single-hop topology (Lille site) (b) Multi-hop topology (Grenoble site)

Figure 2.2: Node deployment in two sites of the FIT IoT-LAB testbed in France.

tocol families NDN, CoAP, and MQTT-SN covering their main variants. To compare producer initiated data transfers in NDN, Interest Notification (I-Not) [14] provides a primitive push mechanism by placing data in an Interest packet, and HoP and Pull (HoPP) [146] serves as a publish-subscribe extension for constrained IoT deployments. We quantify the performance of information-centric protocols and end-to-end approaches, and implemented characteristic IoT use cases for ten variants of these protocols. We deployed them in single-hop (Figure 2.2a) and multi-hop topologies (Figure 2.2b) on the large-scale FIT-IoT LAB testbed, and ran competitive performance contests under fully equivalent conditions.

## 2.2 Media Access in Wireless ICN Networks

**Broadcast vs Unicast.** NDN has the potential to improve performance of application scenarios that connect devices via lossy media such as low-power radios (*cf.* Section 2.1). Caching on the network layer helps to compensate interference on the data link layer by placing contents closer to the sink, which leads to reduced hop-counts and thus to reduce packet loss on the application layer. In contrast to the IP-stack, however, there is no clear mapping between a content name and the neighbor MAC address.

Broadcasting on the data link layer simplifies content sharing in NDN and adds redundancy. Not all nodes in a constrained IoT network can provide caching services, however, because of low-end hardware capabilities and limited memory resources [307]. Hence, broadcast does not immanently lead to content distribution. Instead, broadcast in the IoT introduces two major drawbacks. First, frames are not filtered by common device drivers of the network interface card (NIC). Due to the broadcast nature of the wireless medium, all frames are processed by the CPU, which conflicts with resource constraints on common IoT nodes, in terms of CPU, memory, and energy. Second, common wireless technologies such as 802.11 and 802.15.4 do not support error handling of broadcast frames (*i.e.,* ARQ). Multicast is not even supported in 802.15.4, the prevalent wireless technology in IoT networks. This imposes significant differences on the data link layer compared to unicast. Mapping data names to unicast links, however, requires basic routing and route maintenance.

Current solutions address these problems only partially. They either implement an NDN-

specific link layer to introduce error-resilience [342, 131, 132, 389] or extend current device drivers to implement name-based filtering on the NIC [343]. A dynamic mapping of unicast MAC addresses to NDN interfaces [361, 33] is not sufficiently explored—nor is the question on how well broadcast can serve ICN needs [219].

This surprisingly unsatisfying state of the art motivates us to revisit the problem and solution space. In Chapter 4 we investigate how the mapping of names to the broadcast and unicast MAC address affects the performance of information-centric networks. In experiments, we conduct network benchmarks on low-end IoT devices to better understand the potentials of a dynamic address-to-face mapping.

**Time Division and Frequency Multiplex.** CSMA/CA media access minimizes wireless interference of simultaneous senders in reach. Without time synchronization, nodes need to keep on the radio at all times in order to receive packets. This is particularly important in receiver-driven networks such as ICN: Producers (sensors) do not know when the Interest request arrives. Keeping the radio and CPU on to be receive-able is one of the most energy consuming tasks of an IoT device and conflicts with constrained energy resources on battery operated nodes. Common countermeasures include duty cycling to enable intermittent sleep cycles which, however, makes these nodes unavailable during power-off. In parallel, the unreliable and fluctuating nature of lossy wireless links affects the transmission reliability, and an increasing number of networked nodes demands for exploiting the frequency spectrum optimally, to avoid wireless interference. In practice, common MAC layers apply time- and frequency multiplexing by coordinating nodes to synchronize to a schedule, and to grant exclusive resource access. The assignment of time slots enables intermittent sleep, while orthogonal frequency slots enable parallel transmissions that exploit the spectrum. Additionally, channel hopping or channel blacklisting minimize the impact of disturbed channels, or temporary fluctuations of a radio channel.The absence of a MAC convergence layer in ICN raises the question of how to transmit Interest and data packets through a coordinated wireless network.

**DSME and LoRa.** Similarly to 6LoWPAN [272] and 6TiSCH [394] for IPv6 networks, ICN benefits from a convergence layer that utilizes the TSCH [174] mode of 802.15.4e [163, 162], which contributes time division and frequency multiplex. We want to enable ICN not only for 802.15.4 networks, but the more challenging long-range radio technology LoRa. The long range fosters wireless interference due to a wide overlap, and slow transmissions occupy the channel for a long duration. To enable time- and frequency multiplex for LoRa, we argue that the Deterministic and Synchronous Multichannel Extension (DSME) [174] MAC layer extension of the 802.15.4e standard is better suited to cope with a long transmission range [11] compared to TSCH. DSME is the more flexible MAC that consists of contention-access and contention-free periods, multiplexed across 16 radio channels. Contrasting TSCH, DSME comprises built-in features such as beacon collision resolution, a slot allocation bitmap to indicate readily allocated or unavailable slots, support for clustered tree topologies, and indirect transmission to cater transmissions of very energy constrained nodes. In Chapter 5 we propose and simulate our

replacement of the de facto LoRa MAC layer as specified by LoRaWAN. Our solution utilizes DSME as a new MAC layer for LoRa and enables information-centric networking, to benefit from built-in ICN features that assist in dealing with the challenging long-range domain.

## 2.3 Long-range ICN and Delay-tolerance at the Edge

**Problems of LoRaWAN.** LoRaWAN is a popular low-power long-range communication system for the IoT, but the network design incurs four notable shortcomings, that prevent an efficient integration into an ICN network [195]: (*i*) LoRaWAN is optimized towards retrieving data *from* constrained nodes. Sending data *to* nodes is expensive and involves significant latencies. Many networks such as the popular community *The Things Network* (TTN) thus deprecate sending data to nodes above a very low message rate, making LoRaWAN unsuitable for most control scenarios. (*ii*) LoRaWAN has not been designed with the objective to provide a platform for Internet protocols. It is possible to use IP and adaptation layers on top of LoRaWAN, albeit very inefficiently. (*iii*) The whole LoRaWAN system is a vertically integrated stack that leads to inflexible system designs. For example, all communication is channeled through gateways as well as application- and network servers that interconnect with applications. (*iv*) The centralization and lock-in to vertical protocol stacks challenge data sharing (between users) and the creation of distributed applications (across LoRa island and the Internet).

**Information-centric System Design for LoRa.** We aim for a better integration of the LoRa-based IoT into the remaining Internet and base our system design on the following four requirements: (*i*) enabling LoRa networks and nodes in these networks to communicate directly with hosts on the Internet; (*ii*) empowering LoRa gateways to act as routers, without the need to employ network servers and to tunnel all traffic to or from them; (*iii*) enabling data sharing and wireless node control; (*iv*) maintaining the important power conservation properties of current LoRaWAN systems. To achieve these goals we replaced the MAC layer with DSME in Section 2.2, and introduce a data-driven layer on top. ICN is well suited for use with LoRa because its hop-wise data replication increases robustness and flexibility while reducing (re-)transmission load. This enhances adaptivity and decreases communication overhead, whereas link capacity is scarce with LoRa. Built-in caches in ICN facilitate more efficient LoRa networks. Requests that are satisfied by an in-network cache reduce link utilization and wireless interference, facilitate node sleep, and potentially reduce long round trips introduced by slow transmissions. But the redesign incurs additional functionality on networked nodes, especially on gateways.

**Delay-tolerance in ICN.** Edge networks, including but not limited to DSME- and LoRa-based networks, introduce variable round trip times (RTTs) on the order of seconds or tens of seconds, due to the underlying power saving regime. This challenges practical ICN-forwarders on the Internet, which operate on a different timescale. In particular, PIT state (*cf.* Section 2.1) expires after a defined timeout value (in NDN, the default Interest lifetime value is 4 seconds) and In-

terest retransmissions, albeit unspecified, commonly align with TCP procedures and implement round trip time estimations. This quickly leads to inefficient polling when sensor data is not (yet) available, and brittleness readjustment of timeout parameters as a consequence of vastly differing RTTs. Furthermore, expired PIT state prevents data forwarding and caching. On the other hand, simply increasing the PIT timeout value is not likely to work well in real-world deployments. A core router might object to spend memory resources, storing many Interests for a very long time. To overcome these challenges, we have developed a delay-tolerant ICN communication framework that allows connecting high latency edge networks, attached via an ICN forwarder on a gateway device, to a "regular" ICN Internet connected via fast wired links. It supports data retrieval with arbitrary orders of delays, and without specific assumptions of typical RTTs on other nodes on the ICN Internet. Thereby, no application awareness is required on gateway nodes. Our framework utilizes ICN-idiomatic communication with slight modifications, to benefit from ICN principles such as accessing named data, Interest/data semantics, caches, aggregation, and flow balance.

We have developed interactions for IoT communication use cases that leverage bespoke capabilities on gateway-based forwarders, which follow two patterns. First, IoT sensor data retrieval from an Internet-based consumer using Interest/data interactions. We leverage the concept of RMI for ICN (RICE [211]) that provides access to static data and dynamic computation results, supporting vastly longer data production and retrieval times. Second, asynchronously "pushing" data from an IoT sensor to an Internet-based consumer with publish-subscribe semantics. We leverage the *reflexive forwarding* extensions for ICN [294] that have been proposed to the Internet Research Task Force (IRTF) for standardization.

**Evaluation Scenarios.** Chapter 5 presents the design of ICN over LoRa, including a suitable DSME configuration and options for mapping ICN messages to DSME. We built a complete simulation environment in OMNeT++. Based on simulation results, we derive preferred mappings and additional node requirements for implementing ICN interaction patterns.

Chapter 6 presents the design of delay-tolerant ICN-interactions and node behavior. We evaluate a complete implementation of the DSME MAC layer for LoRa and the ICN protocol extensions on RIOT, serving common LoRa sensors and RIOT-based gateways. We perform experiments of the interactions on off-the-shelf IoT platforms, connected to an emulated ICN Internet, and provide a comparison with unchanged NDN interactions.

# Chapter 3

# Potentials of ICN for Constrained IoT Networks


## Abstract

In this chapter, we start from two observations. First, many application scenarios that benefit from ICN involve battery driven nodes connected via shared media. Second, current link layer technologies are completely ICN agnostic, which prevents filtering of ICN packets at the device driver level. Consequently, any ICN packet, Interest as well as Data, is processed by the CPU. This sacrifices local system resources and disregards link layer support functions such as wireless retransmission. We argue for a mapping of names to MAC addresses to efficiently handle ICN packets, and explore dynamic face-based mapping schemes. We analyze the impact of this link layer adaptation in real-world experiments and quantitatively compare different configurations. Our findings on resource consumption, and reliability on constrained devices indicate significant gains in larger networks.


## 3.1 Background and Use Cases

### 3.1.1 CoAP

CoAP, the Constrained Application Protocol [341], was designed to support REST services in machine to machine communication. Basically, it aims for replacing HTTP on constrained nodes. In contrast to HTTP, CoAP is able to run on top of UDP and introduces a lean transactional messaging layer to compensate for the connectionless transport. CoAP provides a more compact header structure than HTTP.

Three communication primitives are currently supported by this extensible protocol: (*i*) pull, (*ii*) push, and (*iii*) observe. Pull implements the common request response communication pattern. However, as IoT scenarios also include the pro-active communication of unscheduled state changes, CoAP was extended to support pushing new events to its peers. Still, this does not allow for publish-subscribe scenarios when producer and consumer are decoupled in time and data is not yet available at the request. The support for delayed data delivery in publish-

subscribe was specified in CoAP observe [167]. Here, clients can signal interest in observing data, which basically means that a CoAP server delivers data as soon as available and maintains state until clients explicitly unsubscribe.

CoAP must be considered as the IETF standard to implement application layer data transfer in the future Internet of Things. Currently, several implementations exist, as well as early adoption in a few selected products and deployments.

### 3.1.2 MQTT

MQTT [36], the Message Queue Telemetry Transport, was designed as a publish-subscribe messaging protocol between clients and brokers. Clients can publish content, subscribe to content, or both. Servers (commonly called *broker*) distribute messages between publishing and subscribing clients. It is worth noting that the protocol is symmetric: Clients as well as brokers can be sender and receiver when MQTT delivers application messages.

MQTT is considered a lightweight protocol for two reasons. First, it provides a lean header structure, which reduces packet parsing and makes it suitable for IoT devices with low energy resources. Second, it is easy to implement. In its simplest form, MQTT offloads reliability support completely onto TCP.

To provide flexible Quality of Service on top of the underlying transport, MQTT defines three QoS levels, which reflect the agreement regarding message transfer between broker and consumer – both can be sender and receiver. *QoS 0* implements unacknowledged data transfer. An MQTT receiver gets a message at most once, depending on the capabilities of the underlying network, as there is no retransmission on the application layer. *QoS 1* guarantees that a message is delivered at least once. This requires that a message is stored at the sender side until an acknowledgement was received. Based on timeouts, an MQTT sender will retransmit application messages when an acknowledgement is missing. *QoS 2* ensures that a message is received exactly once, to avoid packet loss or processing of duplicates at the MQTT receiver side. This requires a two-step acknowledgement process and more states at both sides.

To adapt MQTT to constrained networks which are based on low data rates and very small packet lengths such as in 802.15.4, MQTT-SN [352] is specified. Header complexity is reduced by replacing topic strings by topic IDs, to identify content. In contrast to MQTT, MQTT-SN is able to run on top of UDP. It still supports all QoS levels but does not inherit any reliability property from the transport layer.

### 3.1.3 ICN Protocols

The core **NDN** protocol [180, 413] combines name-based routing from TRIAD [135] and stateful forwarding from DONA [208] to implement a request response scheme on the network layer. Any consumer can request data that is subsequently delivered along a trail of reverse path forwarding states. As an important feature, data will only be delivered to those who requested

Table 3.1: Comparison of CoAP, MQTT, and ICN protocols. CoAP and MQTT support reliability only in confirmable mode (c) and QoS levels 1 and 2 (Q1, Q2).

| | **Current IoT Protocols** | | | | | **ICN Protocols** | | |
|---|---|---|---|---|---|---|---|---|
| | CoAP [341] | | | MQTT [36] | MQTT-SN [352] | NDN [180, 413] | I-Not [14] | HoPP [146] |
| | PUT | GET | Observe | | | | | |
| Transport | UDP | UDP | UDP | TCP | UDP | n/a | n/a | n/a |
| Pub/Sub | ✘ | ✘ | ✓ | ✓ | ✓ | ✘ | ✘ | ✓ |
| Push | ✓ | ✘ | ✓ | ✓ | ✓ | ✘ | ✓ | ✘ |
| Pull | ✘ | ✓ | ✘ | ✘ | ✘ | ✓ | ✘ | ✓ |
| Flow Control | ✘ | ✘ | ✘ | ✓ | ✘ | ✓ | ✘ | ✓ |
| Reliability | (c) | (c) | ✘ | (Q1, Q2) | (Q1, Q2) | ✓ | ✓ | ✓ |

the data. This means that data must be (individually) named at the Interest request and that yet unavailable data requires repeating Interests until the application receives the data.

The lack of push primitives in NDN triggered the idea of inverting the NDN semantic by placing data in an **Interest Notification (I-Not)** which in turn gets acknowledged by the subsequent (empty) data packet. This idea was originally proposed in [14] and was since then criticized for its lack of (*i*) caching support, (*ii*) flow control, and (*iii*) DDoS resilience.

Several publish-subscribe extensions have been proposed for NDN (COPSS [69], PSync [414]) to provide further decoupling of consumers and data sources. As COPSS relies on a persistent forwarding infrastructure and PSync on Interest broadcasting, both schemes do not satisfy the requirements of the constrained IoT. Our lightweight IoT variant **HoP and Pull (HoPP)** [146] provides a publish-subscribe system for constrained IoT deployments based on ICN/NDN principles. A constrained IoT publisher announces a name towards a content proxy to trigger content requests and to replicate the data towards a content proxy (or broker). Forwarding nodes on the path between publisher and content proxy hop-wise request content for this name by using common Interest and data messages. A content subscriber in HoPP behaves almost like any content requester in NDN and issues a regular Interest request towards the content proxy CP. However, in contrast to NDN (*i*) a subscriber cannot extract content names from its FIB, since FIBs only contain PANINI default routes [330], but uses application-specific topic tables instead; (*ii*) it does not expect an immediate reply, but issues Interests with extended lifetimes. HoPP enables rapid communication of unscheduled data events. It operates at a similar timescale as push protocols without actually pushing data.

### 3.1.4 Protocol Comparison

Key properties of the three protocol families NDN, CoAP, and MQTT and its variants are compared in Table 3.1. Specialized properties of the different approaches become apparent: Every protocol variant features distinct capabilities. Notably in the IoT, where TCP (aka

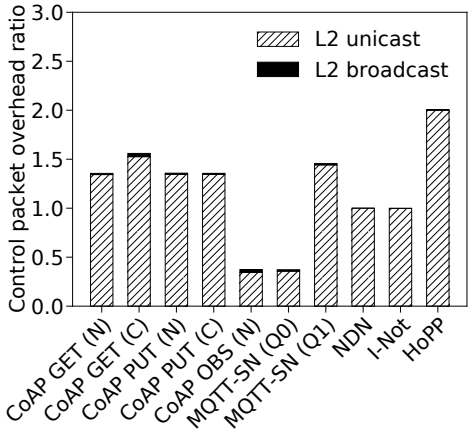



Figure 3.1: Relative protocol overhead under relaxed network conditions incl. topology control broadcasts.

generic MQTT) is unavailable, the pull-based NDN and NDN-HoPP are the only protocols admitting flow control and reliability as a generic service.

An additional mechanism for link recovery and retransmission has been brought to NDN with NDNLP [342]. Facing the lossy nature of low-power wireless links in the IoT, it may be tempting to deploy this additional protocol to enhance the overall reliability. However, common radio links like IEEE 802.15.4 already feature ARQs (Automatic Repeat Requests), and a network layer link would put a second acknowledgement to the air, which in turn would increase the omnipresent risk of interference. For this reason, we did neither deploy nor further investigate NDNLP in our further analyses.

Figure 3.1 compares the control overhead for all protocol variants under consideration as obtained from experiments under relaxed network conditions at negligible interference. Aside from topology building and maintenance that are mainly broadcasts (marked in Fig. 3.1), common request protocols require one request per data item, whereas publish-subscribe schemes only require subscription notification per topic. As a pull protocol, HoPP requires requests and an additional message to advertise names.

Common IoT deployment use cases consist of stub networks as visualized in Figure 3.2 that may be single- or multi-hop. Traffic flows from or to the IoT edge nodes in three patterns: (*i*) scheduled periodic sensor readings, (*ii*) unscheduled and uncoordinated data updates, or (*iii*) on demand notifications or alerting. It is worth noting that the different protocol properties (e.g., push versus pull versus pub-sub) can serve these alternating needs in a quite distinct manner.

## 3.2 Implementation and Experimental Setup

**Software Platforms.** On the IoT nodes, all of our experiments are based on RIOT version `2018.01`. To analyze CoAP, MQTT-SN, and NDN we use `gCoAP`, `Asymcute`, and `CCN-lite`

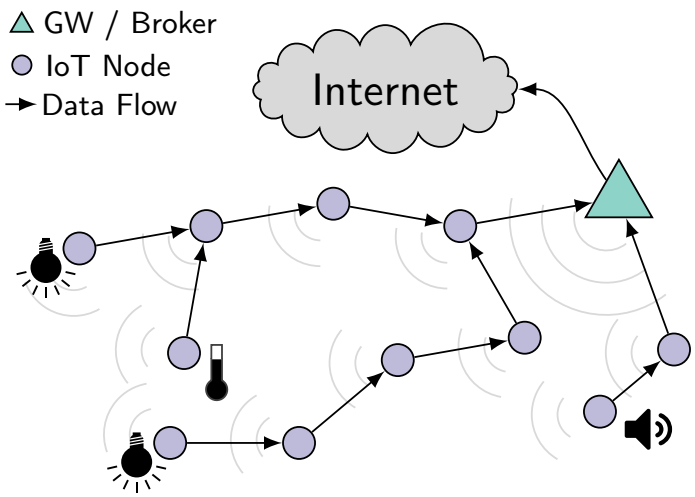



Figure 3.2: Use case scenario of a multi-hop IoT topology.

respectively. All three protocol implementations are part of the common RIOT release and thus reflect typical software components used in low-end IoT scenarios.

On the brokers or gateways, the testbed infrastructure deploys Linux systems. To support MQTT broker and CoAP observe client as well as CoAP PUT server functionalities, we used `aiocoap` version 0.3 and `mosquitto.rsmb` version 1.3.0.2. Both are popular open source implementations in this context.

**Testbed.** We conduct our experiments in the FIT IoT-LAB testbed. The hardware platform consists of typical class 2 devices [58] and features an ARM Cortex-M3 MCU with 64 kB of RAM and 512 kB of ROM. Each device is equipped with an Atmel AT86RF231 [29] transceiver to operate on the IEEE 802.15.4 radio. The gateway runs on a Cortex-A8 node, which is more powerful than the M3 edge nodes.

The testbed provides access to several sites with varying properties. We perform our experiments on two sites, to analyze single-hop as well as multi-hop scenarios.

**Single-hop topology** The *Paris* site consists of approximately 70 nodes, which are within the same radio range. We choose two arbitrary nodes and run all single-hop experiments on them. One node is a content producer, the other node acts as consumer (gateway/broker).

**Multi-hop topology** The *Grenoble* site consists of approximately 350 nodes spread evenly in the Inria Grenoble building. We choose 50 M3 nodes (low-end IoT device) and one A8 node (gateway/broker) arbitrarily and run all multi-hop experiments on them. All low-end devices operate as content producers. In our CoAP and MQTT experiments, we use RPL to build and maintain the routing topology across all nodes. In our NDN-based experiments, we build tree topologies analogously as HoPP does. In any case, we ensure that all protocols use the same routing topology for comparison. Typical path length are four to five hops.

**Scenarios and Parameters.** We align all experiments with respect to the configurations of retransmissions and timeouts to ensure comparability among protocols. All protocols employ the same retransmission strategy: In case of failures, each node waits 2 seconds before retransmitting the original application or control data. For NDN, HoPP and I-Not, retransmissions are performed hop-by-hop, while CoAP and MQTT perform them end-to-end. At most 4 retransmissions will occur for each data. Interest lifetimes are configured to 10 seconds for NDN based protocols to limit PIT memory consumption. We repeat each experiment 1,000 times.

To accommodate all 50 nodes in the routing topology, the FIB size was adjusted accordingly on each IoT node. For CoAP and MQTT, this translates in our IPv6 scenario to a FIB size of 50 entries with roughly 32 bytes each (`sizeof(destination)` + `sizeof(next-hop)`). In our NDN scenarios, each node owns a unique prefix of the form $/\rho_i$ with a length of `24` bytes. The next-hop face of each FIB entry points to the `8`-byte IEEE 802.15.4 link layer address. In total, this setup yields comparable size requirements for all scenarios.

In the NDN scenarios, we use unique content names prefixed by $/\rho_i$ with incremental local packet counters. CoAP works without unique names but uses common URIs. The MQTT-SN protocols register a common topic name, similar to CoAP, and publish under a unique topic ID thereafter. In all scenarios, the data is of the same JSON format consisting of a unique identifier and a sensor value attribute. These short messages can be accommodated by the link layer and do not require fragmentation. It is noteworthy that we neither applied header compression in the IP [57] nor in the NDN world [161].

## 3.3 Evaluation

### 3.3.1 Analyses and Metrics

The objective of this work is to quantify the efficiency and utility of the considered protocols in real deployment scenarios. With this in mind, we want to shed light on resource consumption and the operational properties of data dissemination from different angles and in the different deployment use cases.

In detail, we analyze the *memory consumption* on nodes, the effective *network utilization* by control and data traffic including *protocol overhead* and *link stress* caused by retransmissions. The actual performance of data transmission is measured in *data loss*, *goodput*, and *content arrival time* which represents the delay between issuing a transaction and data arrival at the sink. Here, we use the term *time to completion* interchangeably. We also consider the *data flows* and its *energy consumption*. These multi-sided analyses are performed on complete packet traces which we recorded from the different experiments, and a monitoring of the system state at participating nodes.

Security measures largely differ between the IP and the ICN world. DTLS [313] provides

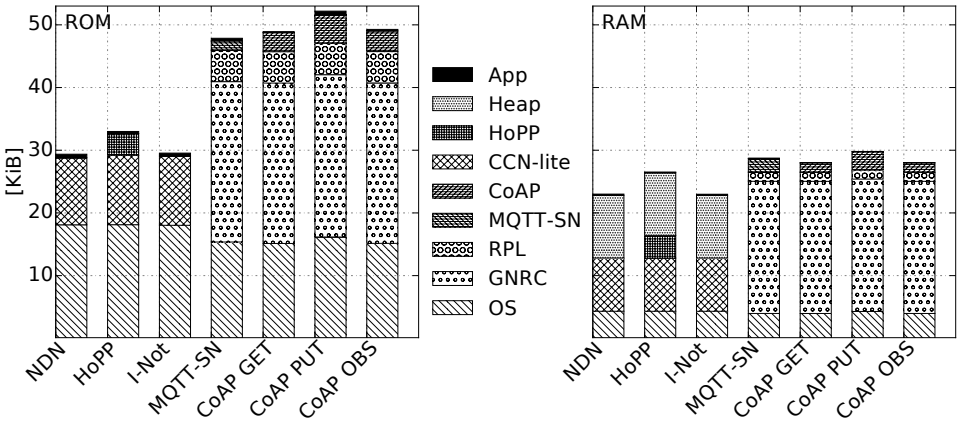



Figure 3.3: Resource consumption of ROM (left) and RAM (right) for the different software stacks.

privacy and integrity for UDP datagrams within sessions based on pre-established private keys. NDN authenticates data chunks between arbitrary endpoints without the need for session state. Canonically, asymmetric signatures are attached to data chunks in NDN, but since the complexity of asymmetric crypto exceeds the capabilities of constrained nodes, keyed-hash message authentication code (HMAC) can also be applied. The use of HMAC likewise relies on pre-established keys.

In both worlds, security extensions add message and processing overhead, but do not change the overall behavior of the protocols. For this reason, we compare security overheads in separate micro-benchmarks and perform the remaining experiments without applying the corresponding security measures.

We do not consider network congestion from external cross-traffic in this work. However, each individual transmission experiences self-induced background traffic from the experiment that differs for varying request/publish intervals and jitter. On average, this side-traffic is constant per experimental run.

### 3.3.2 Protocol Stack Sizes

Largely differing properties and complexities of the protocol variants under test lead to seven distinct software stacks. Nodal memory consumption for these different protocol stacks are depicted in Figure 3.3. We differentiate the protocol layers in place to disclose the details.

Main memory is the scarcest resource in the IoT. While protocols require OS support of 4,060 B (MQTT-SN) – 4,400 B (NDN) kernels, NDN admits the leanest stack of 8,700 B consumed by CCN-lite. All IP protocol stacks are significantly larger and approximately triple the size of CCN-lite. On the overall, about 30 KiB are needed to host IP protocols, leaving only a few dozen KiBs for the application on typical constrained nodes. All ICN protocols provide a Content Store (CS) of 10,240 B on the heap, which is the price of in-network caching. It

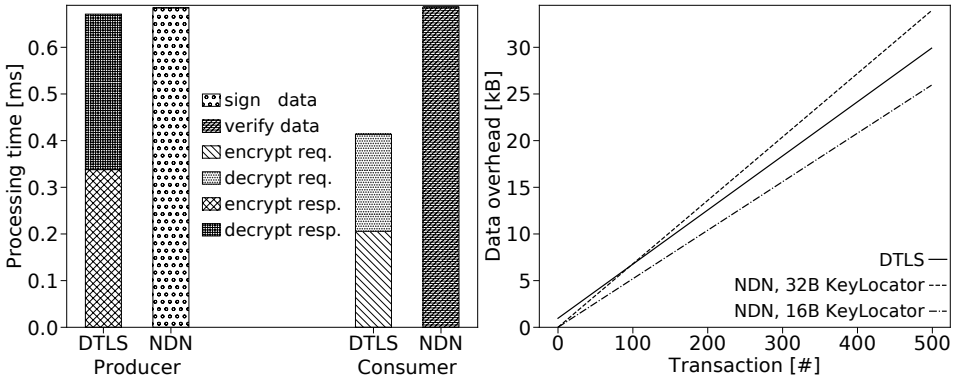



Figure 3.4: Security overheads—CPU consumption (left) and data overhead (right) per content transaction for IP/DTLS and NDN/HMAC.

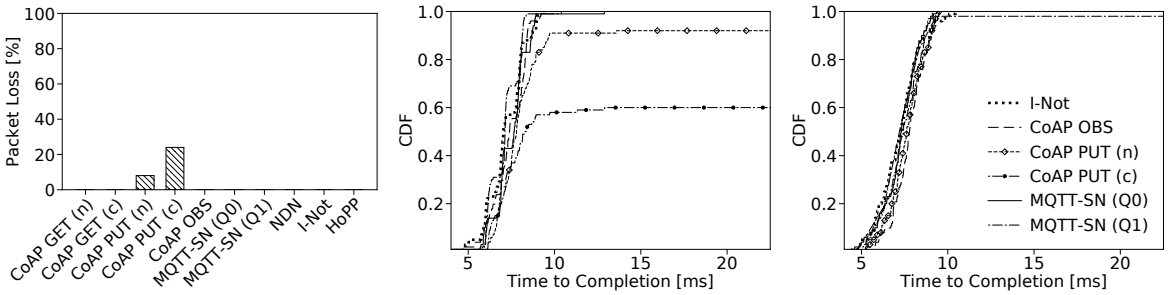
should be noted that the GNRC network stack contributes a packet buffer to both, the IP and the ICN world that is also used for retransmissions [230]. Program sizes of NDN protocols are much smaller and consume about 40 % less ROM. The operating system support varies with protocol requirements on the highly modular RIOT OS platform.

### 3.3.3 Security Overheads

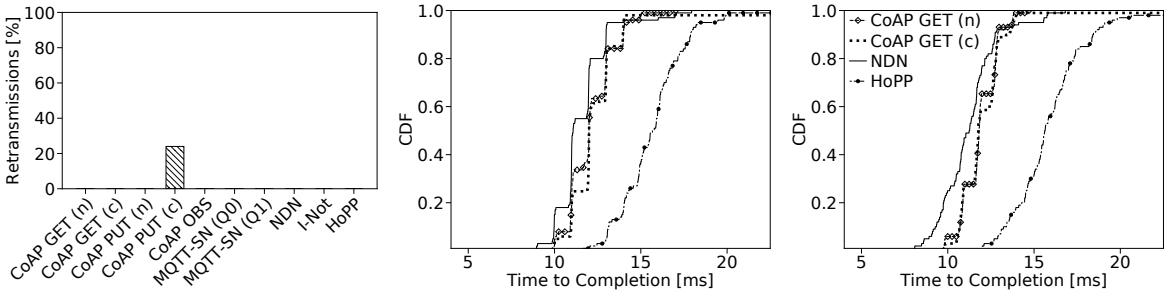
Many use cases of the IoT rely on integrity and authenticity of the collected data. Security extensions of the communication protocols are requested to ensure those properties at costs which we are now evaluating. For our micro-benchmarks of the IP world, we fixed the scenario of a DTLS session established between two nodes. We quantify the messaging overhead obtained from a single session establishment and the packet overhead as a function of data transactions—the request/response-guided transfer of a data unit. We also recorded the CPU expenses at the content producer and consumer per transaction.

The most comparable scenario for NDN consists of HMAC-based authentication of data using SHA256 per chunk. For quantifying the overheads in data packets, we chose two common sizes of the KeyLocatorTLV: 16B and 32B.

Figure 3.4 visualizes the results of our security benchmarks performed on the IoT-LAB M3 nodes. While message overheads for NDN are similar or better (for 16B KeyLocatorTLV), DTLS data verification can be performed at two-thirds of the NDN costs. It should be noted, though, that the different security models of DTLS and NDN make comparisons difficult. While DTLS operates within an established session that is strictly bound to endpoints, the content of NDN can be replicated between varying nodes. In particular, the NDN approach is robust under mobility and network changes, whereas DTLS would require to re-establish sessions in many cases at significant cost. Conversely, only DTLS encrypts transport payloads and thereby contributes data privacy.




(a) Packet loss at 50 ms publishing interval (b) Push at 50 ms publishing interval (c) Push at 5 s publishing interval




(d) L3 retransmissions at 50 ms interval (e) Pull at 50 ms publishing interval (f) Pull at 5 s publishing interval

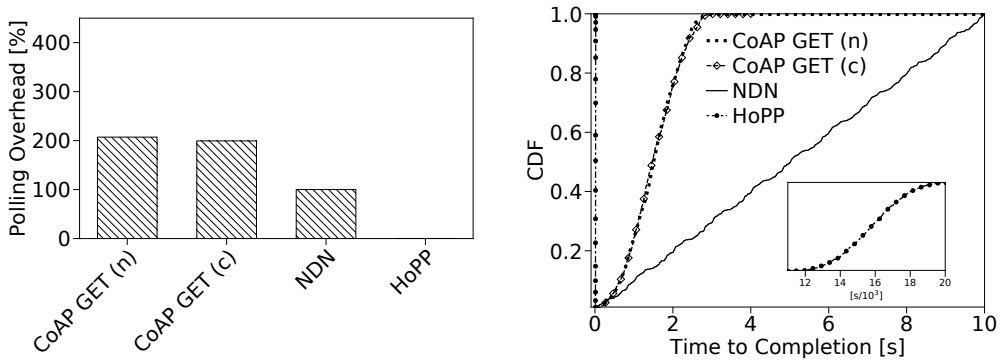
Figure 3.5: Time to content arrival for scheduled publishing in a single-hop topology at different intervals.

### 3.3.4 Single-hop with Scheduled Publishing

Protocol performances are first evaluated in a single-hop topology at the Paris testbed with periodic content publishing every 50 *ms* and 5 *s*. Content is pushed or requested accordingly. Figure 3.5 displays the results for protocol reliability and temporal performance. As an overall trend, it is apparent that push-oriented protocols operate faster, but less reliable.

For the rather relaxed scenario of publishing every 5 *s*, we see the protocol families in rough agreement. Push-based protocols require an average of 7 *ms* (Fig. 3.5c) for data delivery, pull-based protocols take 11 *ms* (Fig. 3.5f), with the exception of HoPP which is slightly slower on this short timescale due to its three-way handshake.

The publishing interval of 50 *ms* puts some protocols under stress, even though IEEE 802.15.4 practically limits transmission only below an interval of 10 *ms*. The performance of CoAP PUT significantly degrades (Fig. 3.5b), leaving the unconfirmed messaging at a total data loss of 6 % (Fig. 3.5a). The PUT of Confirmable CoAP instead initiates 26 % retransmissions (Fig. 3.5d) which increase delays up to a factor of five. Confirmable CoAP does complete data delivery at 42 *ms* (Fig. 3.5b is clipped for visibility). It should be noted, though, that retransmissions on the data link layer are present for all protocols and are reflected by the staircase patterns. We




(a) Control Overhead for polling unscheduled content

(b) Time to unscheduled content arrival

Figure 3.6: Pull protocol performance at random publishing in [1s ... 3s].

do not measure these fast repeats ($\leq 10\ ms$) in this work, but refer to our previous study [193] for further details.

### 3.3.5 Single-hop with Unscheduled Publishing

Our next experiments address the common IoT use case of publishing data at irregular intervals. This is the typical pattern for observing third party actions (*e.g.,* light switching), or largely uncoordinated sensing environments. Push-based protocols naturally serve these application needs. We quantify the behavior of the request-based protocols in practice and chose the moderate setting of publishing content every two seconds on average. Publishing is uniformly distributed in the interval of [1 s ... 3 s]. The protocols CoAP and NDN request the content periodically every second so that updates are not lost.

Figure 3.6b visualizes content delivery times for all request-oriented protocols. CoAP GET and NDN now operate on a timescale of seconds, while HoPP continues to complete in the unaltered range of 15 *ms* without additional protocol operations – the unsurprising outcome of content triggers built into HoPP. CoAP requests content using a common name with the result of likely duplicate content transmissions. On average, CoAP needs two requests to retrieve fresh content with the expected average delay of $\approx 2\ s$ and a corresponding polling overhead of 200 % (Fig. 3.6). In contrast, NDN admits lower overhead, as Interests are locally managed at the PIT and only retransmitted after state timeout.

However, issuing Interests at a higher rate than content arrival leads to an accumulation of open states in the PIT. As resources on the constrained nodes are tightly bound, the PIT limits are quickly reached and can be only met by either *discarding* newly arriving Interests, or by *overwriting pending Interest state*. Both countermeasures delay content delivery, as can be seen from Figure 3.6b. In detail, the time to content delivery of NDN stretches over various PIT combinations up to the final PIT timeout of 10 *s*. It is noteworthy that PIT overflow

in these experiments appears for available content that is ready for delivery via valid routes. NDN protocol extensions such as NACKs would neither help nor should be triggered, since Interest retransmissions act as counter measures to packet loss or timeouts due to wireless link degradation. Consequently, the quantitative impairment of packet delivery tightly depends on the scenario and can lead to significant data loss in the constrained IoT, as well.

These experiments shed again light on the trade-off between memory and network performance in the NDN stateful forwarding regime as has been first identified in [386] and recently discussed in the IoT context [339].

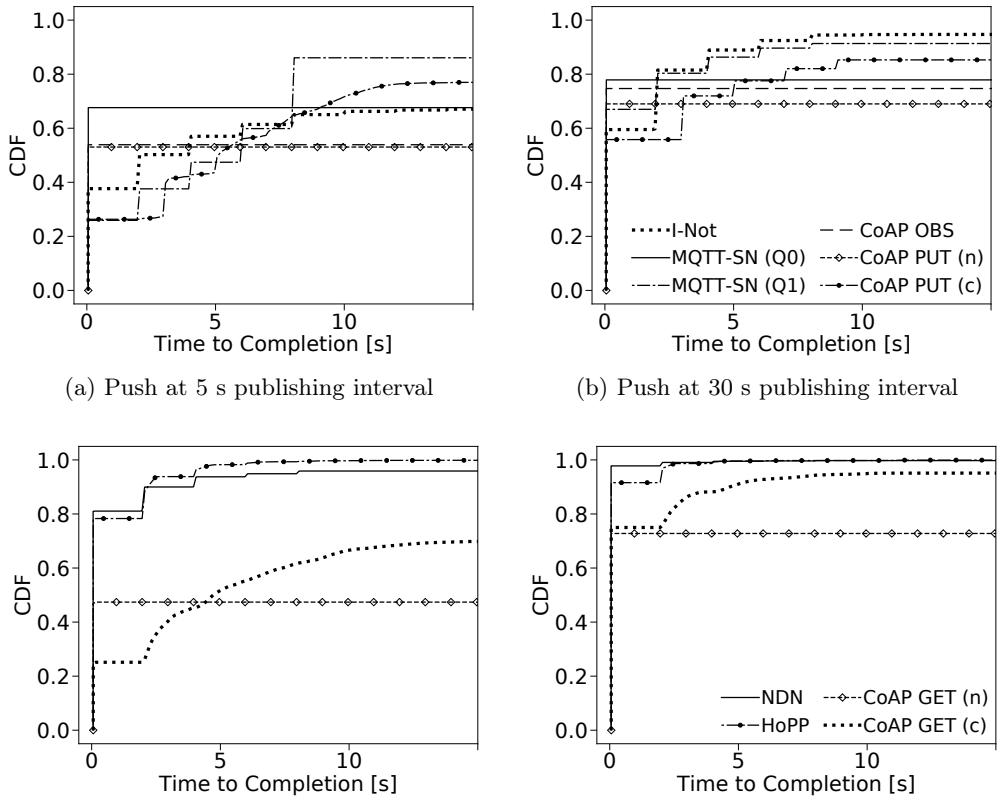
### 3.3.6 Multi-hop Topologies

We now consider the more delicate use case of mixed communication in multi-hop topologies: 50 nodes exchange content that is published every 5 or 30 seconds in an uncoordinated manner. Repeated experiments were performed on the Grenoble testbed with tree topologies of routing depths varying from four to six hops.

First, we examine the temporal distributions from content publishing to arrival in analogy to the single-hop cases. Figure 3.7 combines the results for push and pull protocols, as well as both publishing rates. The overall results reveal a much slower and less reliable protocol behavior than could be expected from the single-hop values in Figure 3.5. Graphs reflect the common experience in low power multi-hop environments that interferences and individual error probabilities accumulate in a destructive manner.

Push and pull protocols now operate on similar time scales in the absence of considerable disturbances, while events of strong interference and packet corruption on the air lead to large retransmission delays and loss. Protocol retransmissions with an interval of 2 seconds are clearly reflected by the staircase patterns in the respective CDF. Most notably, the 'reliable' variants of CoAP PUT (c) and GET (c) fail to always transfer the content, but remain unsuccessful in a range between 5 % (at 30 s publishing) and 30 % (at 5 s). Even though confirmable CoAP operates more reliably than the unreliable versions OBS and PUT/GET (n), the failure rates indicate a quite unsatisfactory protocol behavior. A similarly unsuccessful performance must be observed for the NDN push variant Interest Notification. In contrast, the reliable MQTT (Q1) successfully transfers its data in 90–95 % of all cases, thereby heavily relying on retransmissions as we will see in the course of the further analysis.

The performance of NDN shows decent results both in promptness and reliability, even though 5 % of data chunks remain lost in the fast publishing scenario (5 s). The only protocol that delivers reasonably fast at full reliability is the NDN variant HoPP. Below we will see that this happens with the least retransmissions and in evenly balanced flows. In a way, this result is not surprising as HoPP is optimized for IoT demands and the only protocol that balances data transmissions per hop. It is the common experience in the low power wireless that link qualities vary quickly and largely.




(a) Push at 5 s publishing interval

(b) Push at 30 s publishing interval

(c) Pull at 5 s publishing interval

(d) Pull at 30 s publishing interval

Figure 3.7: Time to content arrival in multi-hop topologies of 50 nodes.

Second, we evaluate the effective data goodput and flow analysis of the different protocols during content publishing experiments. In Figures 3.8 and 3.9, we summarize the results for the variants of NDN, MQTT, and CoAP respectively. We display the different experimental results of the data goodput in box plots and compare to the theoretical optimum (lines). Time series of data goodput are further revealing the flow behavior as displayed in the lower row of these figures.

Clearly, HoPP admits the most evenly balanced flows and shows nearly optimal goodput values, closely followed by NDN. All other flow performances fluctuate with some tendency of instability when approaching its full transmission speed. Some IP-based flows in MQTT and CoAP drop to lower delivery rates which is dominantly caused by slow repeated end-to-end retransmission. Multi-hop retransmissions in this error-prone regime tend to cause additional interferences and accumulate transmission errors. As a consequence, protocols operate at reduced efficiency – in some cases protocol performance drops down to 50 % (*e.g.,* CoAP GET (n) and CoAP OBS in Fig. 3.9). Interest Notification which is not capable of content caching, does not outperform the IP protocols. The overall results show that the absence of flow control as in UDP/IP–based protocols and in the I-Not variant of NDN make protocols fragile. Hop-wise

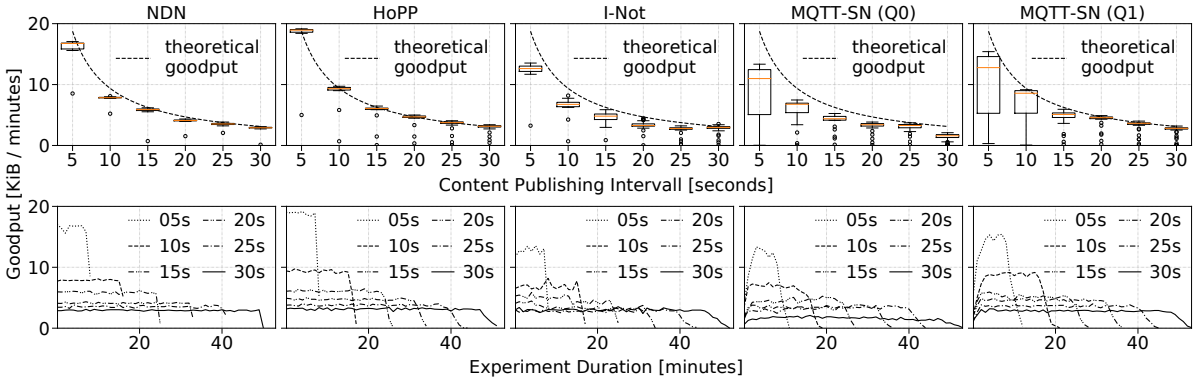



Figure 3.8: Goodput summary and evolution for the NDN and MQTT protocols at different publishing intervals.

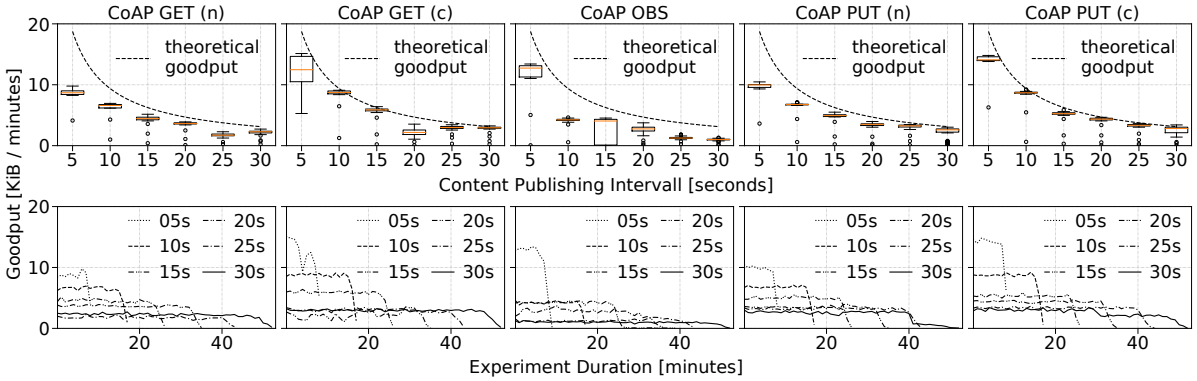



Figure 3.9: Goodput summary and evolution for the CoAP protocols at different publishing intervals.

retransmission management as applicable in NDN and HoPP re-balances flows and explicitly demonstrates its benefits for the IoT instead.

Our next evaluation focuses on the link utilization. We measure all individual paths that each unique data packet traveled on its destination from source to sink and contrast the results with the corresponding shortest possible path. Results are visualized as scatterplots in Figure 3.10. Each dot represents one or several events, the dot size is drawn proportionally to event multiplicities. Solid lines indicate the shortest paths, while events left of the line represent failures (traversal shorter than the shortest path). Right of the solid diagonal retransmissions are counted.

The ideal protocol performance is situated on the diagonal line with all data traversing each link only once on the shortest path. This ideal behavior is most closely approximated by the NDN core and the NDN HoPP protocols. A largely contrasting performance can be seen from the reliable IP protocols MQTT (Q1) and CoAP PUT (c) which admit huge numbers of

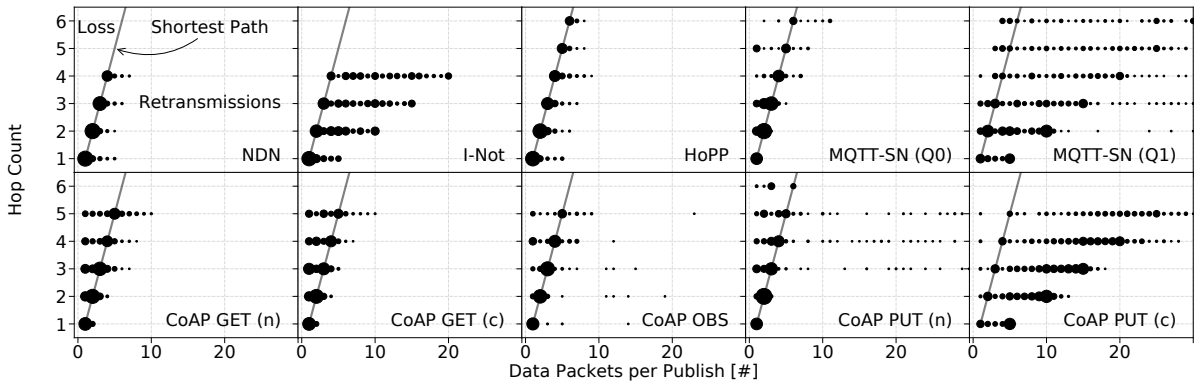



Figure 3.10: Link traversal vers. shortest path for a 15 s publishing interval. The scatterplots reveal the link stress with dot sizes proportional to event multiplicity.

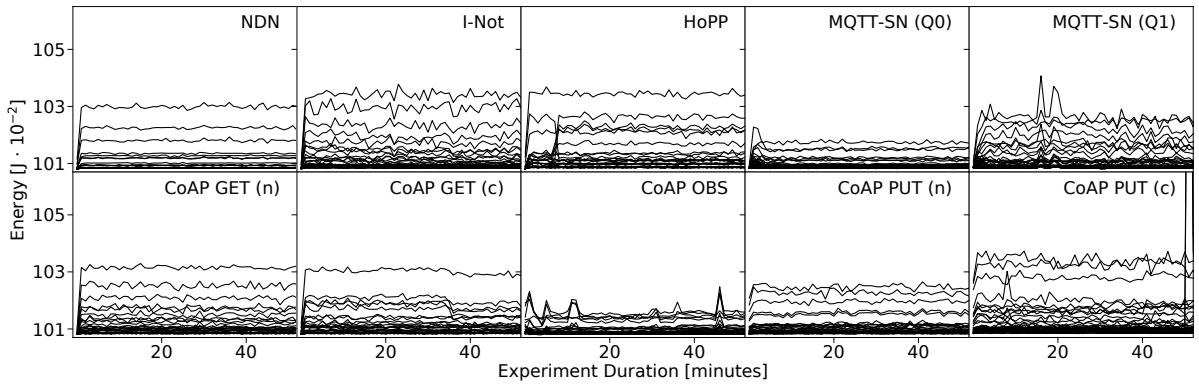



Figure 3.11: Energy consumption over time for each node in the topology using a 15 s publishing interval.

retransmissions. This also holds for the NDN Interest Notification protocol which cuts out the NDN feature strength by inverting its semantic.

Unreliable IP-based protocols show very large loss multiplicities and only a few retransmissions which are initiated by reacting to link-layer failures. This corresponds to the reduced success rate already observed in the previous evaluations. Apparently, all protocols that follow an end-to-end path semantic (including I-Not) are forced to struggle against the unpredictable nature of intermediate links—either by voluminous packet retransmissions or significant packet loss.

In our final experimental comparison between the protocols, we evaluate the individual energy consumption per node as a function of time. Since the energy demand of a protocol is largely dominated by its radio transfer of packets, we focus our measurements on 'bytes in the air', i.e., the IEEE 802.15.4 transmission and reception of packets on each individual node. Power consumption levels for transmitting, receiving and idling are obtained from the Atmel AT86RF231 data sheet, and we calculate the actual energy from measuring the radio operation time in the respective device state.

Time series of nodal energies are plotted in Figure 3.11 for each protocol during the course of the experiment. Immediately we observe the tree topology pattern in all graphs: The root node prominently consumes a multiple of leaf node energies, and intermediate forwarders differentiate according to tree ranks in between. It is noteworthy that the routing topology did not rearrange during the measurement period. A varying use of routing trees could gradually balance the uneven energy needs.

Aside from topological effects, distinct protocol signatures become visible. While all energy curves fluctuate due to temporal variations and local retransmissions, some protocols show significant amplitudes from local disorder and repair. Reliable MQTT (Q1) exhibits a peak of recovery after an initial period of loss with depleted energy level on some branch, and a high number of pronounced peaks otherwise.

HoPP experiences a handover in energy load at about eight minutes. This follows its ability of dynamically switching to a more reliable uplink path without rebuilding the topology. HoPP and NDN admit rather steady and smooth energy gradients, since they mainly rely on local repairs (or caching). In contrast, I-Not as a protocol without in-network caching support requires more hop-wise retransmissions and must be considered energy-wise expensive.

Unreliable protocols such as MQTT (Q0) and CoAP (n) repeatedly show valleys in energy curves, since packets lost early on the path relieve the burden of forwarding to the remainders. Delivery failures in CoAP GET (c) as already known from Figure 3.10 lead to some drops in energy, as well. MQTT (Q0), CoAP OBS and CoAP PUT (n) consume the least energy, which is not surprising for these lean protocols without loss recovery.

Viewing link-stress (Fig. 3.10) and energy flow (Fig. 3.11) conjointly, a rather clarifying view on the operational conditions of the protocols emerges. Some protocols remain lean and undemanding while delivering only a restricted service (e.g., CoAP OBS and PUT (n)), others are steady, predictable and run at full service (e.g., NDN and HoPP), and some protocols really struggle in this IoT-typic environment (e.g., MQTT (Q1), CoAP PUT (c), and I-Not).

## 3.4 Related Work

### 3.4.1 ICN and IoT

The benefits of ICN/NDN in the IoT have been analyzed mainly from three angles. (*i*) design aspects [33, 307, 337, 258, 27], (*ii*) architecture work [120, 331], and (*iii*) use cases [62, 15, 325, 141]. To support experimental evaluation, several implementations have become available, including CCN-lite [372] on RIOT [32, 34] and on Contiki [8], and NDN on RIOT [338]. The objective of this chapter is not to present an additional ICN implementation for the IoT but to reuse common stacks. With this we contribute to more reliability of existing software as extensive usage helps to find bugs.

The evaluation of NDN protocol properties in the wild includes the exploitation of NDN com-

munication patterns to improve wireless channel management [162, 164] as well as data delivery on the network layer [33]. Comparison to common IoT network stacks at the transport layer (in particular UDP) is not available. In this chapter, we close the gap towards the application layer and analyze common application protocols (*i.e.,* MQTT and CoAP) compared to intrinsic network layer characteristic provided by NDN.

### 3.4.2 Interoperation and Adoption of CoAP and MQTT in ICN

Implementing CoAP on top of ICN has been proposed to enable full features of CoAP [113, 358], such as support of group communication and delay-tolerant communication [179]. These concepts have been showcased in building management systems [114]. In contrast to the integration of CoAP into ICN, an MQTT-to-CCN gateway was proposed to allow for interoperation between CCN IoT devices and the current Internet [8]. A dedicated rendezvous point to discover resources and to bridge between IP-based MQTT subscribers and NDN sensors was introduced in [181]. Note that our work differs from those research as we assess the performance of CoAP, MQTT, and NDN in their original deployment scenarios, instead of focusing on interoperability use cases. This helps to identify intrinsic protocol characteristics.

### 3.4.3 Performance evaluation of CoAP and MQTT

The performances of CoAP and MQTT have been studied from several perspectives over the last years [175, 89]. Very early work analyzed the interoperability of specific CoAP implementations [234, 381] without performance evaluation. Later, CoAP implementations have been assessed in comparison to HTTP [246] or on different hardware architectures [212]. MQTT was evaluated in [103]. Thangavel *et al.* [363] proposed a common middleware to abstract from CoAP and MQTT. Based on this middleware, CoAP and MQTT were evaluated in a single-hop wired setup. In emulation, MQTT and CoAP have been studied in the context of medical application scenario [70]. A holistic analysis of MQTT and CoAP in a consistent experimental setting including low-end IoT devices is missing. In particular, no detailed comparison to NDN is provided.

## 3.5 Conclusions

This chapter presented extensive experimental analyses to answer the question which of the common protocols MQTT, CoAP, or NDN is best suited for transferring IoT data from constrained devices. We found that for simple, single-hop topologies the protocol families examined in this chapter behave similar, but lean push protocols such as MQTT-SN and CoAP Observe operate fastest, at lowest energy consumption, and most network-friendly.

In challenged multi-hop scenarios, though, the results quickly turn tides into a differentiated view between protocols that operate in host-to-host semantic and those acting per link traversal.

NDN and NDN-HoPP can both enfold their hop-wise transfer features in balancing flows that reliably deliver data without the need for remarkable retransmission rates. This is in significant contrast compared to common UDP-based IoT application layer protocols that do not benefit from underlying flow control.

While NDN is susceptible of overflowing PIT states in unscheduled publishing scenarios, NDN-HoPP handles such notification events without any performance flaw. In contrast, all IP-based protocols and also the NDN Interest Notification quickly struggle in challenging regimes, either by losing or by repeating packets at large scale.

Our overall findings clearly indicate that lean and simple protocols such as MQTT and CoAP Observe can enfold its efficiencies in relaxed environments with low error rates. Challenged networks, though, will quickly degrade their performance to a minimum. In disruptive environments, protocol performance improves with operations confined to the local link: Hop-by-hop transfer with intermediate caching notably increases reliability and reduces corrective actions, which jointly grants efficient robustness. Dependable systems in challenged regimes should take advantage of corresponding solutions.

# Chapter 4

# MAC Address Mapping in ICN


## Abstract

In this chapter, we start from two observations. First, many application scenarios that benefit from ICN involve battery driven nodes connected via shared media. Second, current link layer technologies are completely ICN agnostic, which prevents filtering of ICN packets at the device driver level. Consequently, any ICN packet, Interest as well as data, is processed by the CPU. This sacrifices local system resources and disregards link layer support functions such as wireless retransmission. We argue for a mapping of names to MAC addresses to efficiently handle ICN packets, and explore dynamic face-based mapping schemes. We analyze the impact of this link layer adaptation in real-world experiments and quantitatively compare different configurations. Our findings on resource consumption, and reliability on constrained devices indicate significant gains in larger networks.


## 4.1 Problem Statement and Related Work

### 4.1.1 The IoT Use Case

The Internet of Things gathers a diverse set of very heterogeneous nodes. Our focus is on low-end IoT devices, equipped with hardware resources of class 2 [58], connected via radio, and powered by battery. These devices benefit from NDN in the following way. First, the lightweight NDN network stack requires less memory compared to the current IoT stack standardized in the IETF. It is worth noting that cache sizes are independent from network stack sizes. NDN provides off the shelf name-based management and monitoring capabilities, without introducing dedicated services on top of the network layer. Second, those devices may offload data to more powerful nodes without the burden of additional protocols. This is implemented natively on the network layer, and thus simplifies application programming. Third, this in-network caching allows nodes to sleep longer, reduce data delivery latency, and increase content availability [164].

It is worth stressing that the IoT does not only gather different nodes among different operators but may also be heterogeneous within a single domain. As such, we cannot assume that all nodes provide the same set of services, neither on the application, nor on the network layer.

### 4.1.2 Current Solutions and Challenges

#### 4.1.2.1 NDN-specific Link Layer

Shi *et al.* [342], Grassi *et al.* [131, 132], and Wang *et al.* [389] argue for a link protocol that is specific to NDN. Shi *et al.* [342] introduce NDNLP, which features fragmentation and reassembly as well as acknowledgement and retransmission of packets. NDNLP is located between the network layer (*e.g.,* NDN) and the virtual (*e.g.,* tunnels) or physical (*e.g.,* Ethernet) link layer. Grassi *et al.* [131, 132] present a link adaptation layer, which is tailored for vehicular networks but follows conceptually the same idea as NDNLP. Similarly, in the context of improved reliability, Wang *et al.* [389] introduce an NDN broadcast protocol, which tries to minimize collision. Both approaches aim for an increased packet delivery ratio by measures below the application and network layer but still require packet processing by the CPU, independent whether a specific NDN service bound to the broadcast packet is available or not.

#### 4.1.2.2 Name-based Filtering on NIC

Shi *et al.* [343] propose name-based filters on the device driver level of the network interface card. To optimize the implementation for limited on-chip memory of the NIC, names are maintained in a Bloom filter table. This approach exhibits good performance results but comes with the drawback of a layer violation. The data structure to implement filtering is specific to the ICN approach above the link layer. However, not all ICN approaches follow the same naming scheme [7, 405]. Consequently, changing the specific ICN network stack may require update of the device driver. This will slow down deployment of upcoming approaches. More importantly, this approach distributes data via layer 2 broadcast frames, and thus does not benefit from error handling on this layer.

#### 4.1.2.3 Unicast Faces

Approaches different from the adaptation of the link layer or device driver are presented by Teubler *et al.* [361] and Baccelli *et al.* [33]. They introduce unicast faces. Basically, unicast faces assign (unicast) MAC addresses to NDN faces. These are created dynamically. Initial Interests are broadcasted, containing the unicast source MAC address of the sender. Having this information in place, the receiver makes use of the source address to assign a unicast face. NDN packets which are transmitted via a unicast face conversely include the unicast MAC address. This allows both native MAC-based filtering and benefiting from error handling/prevention on the data link layer. This approach is suitable for specific adaptations to link layers like TSCH [162], and in case not all nodes within a broadcast domain provide the same network layer services, such as in the IoT. On the other hand, unicast traffic reduces caching capabilities and data redundancy. A detailed analysis of link layer unicast and broadcast on the system load of an NDN node is still not present. In this chapter, we argue that NDN should revisit the MAC

layer mapping. There are application scenarios in which a reduced system load outperforms data redundancy. Our analysis in Section 4.3 is a first step in this direction.

## 4.2 Design Space by Instrumenting Existing Link Layer Features

An NDN node can send Interest as well as data packets via unicast or broadcast on the MAC layer. In this section, we discuss pros and cons of each configuration scheme and perform a first experimental reality check about the effect of link layer support.

For the sake of clarity, we focus on core aspects. We assume NDN nodes with a single interface connected to the network via shared media. Extending this scenario to nodes with multiple interfaces does not change the core insights. Furthermore, we do not explicitly discuss multicast for two reasons. First, typical lower layer IoT protocols (*e.g.,* 802.15.4) do not support multicast. Second, linking a face to a multicast MAC address instead of a unicast MAC address requires only a name to group address mapping on the data link layer.

### 4.2.1 Broadcast or Unicast for Interest or Data

**Case 1: Interest Broadcast, Data Broadcast.** Any node within the broadcast domain will send Interests as well as data as link layer broadcast. As long as there is a matching name prefix in the forwarding information base (FIB), these (successfully received) Interests will create a PIT entry. Consequently, as soon as a corresponding data packet is transmitted within the broadcast domain, all members of this domain will forward this data packet, leading to redundant traffic. This highest level of redundancy has pros and cons. Practically, in a densely connected network any node may fail without degrading data delivery, as all remaining nodes cache content. However, this level of packet redundancy introduces excessive overhead for each node (*e.g.,* CPU processing) but also for the complete network (*e.g.,* radio interferences). Interferences should be considered even more seriously with broadcast traffic, as there is no protective repair of errors on the link layer.

**Case 2: Interest Broadcast, Data Unicast.** Similar to case 1, all nodes of the broadcast domain will create a PIT entry after receiving an Interest packet. However, data packets are directed to the unicast MAC address which is associated with the corresponding outgoing face (*i.e.,* the MAC address of the next hop). As we assume a shared media, all other nodes within radio reach receive the data packet as well, but drop it at the device driver level because of an unmatching (unicast) MAC address. Those nodes can neither cache nor forward the data on the network layer. Previously created PIT entries will thus not dissolve by receiving data but by timeouts. These PIT entries require memory, processing time and will not help to achieve redundancy. To cope with node failures, an additional mechanism is needed to keep the MAC-face assignment in sync with the MAC address of an alternative next hop.

**Case 3: Interest Unicast, Data Unicast.** Compared to the previous scenarios, in this

case, Interest as well as data packets are sent to a unicast MAC address, using unicast faces as described in Section 4.1.2.3. Such an approach implements hop-by-hop forwarding on the link layer and prevents redundancy completely because any overheard packet is dropped by the network interface card. This setup requires active maintenance of MAC-to-face mapping in case of node failures.

In contrast to Case 2, updating only the unicast data face is not sufficient. Data will be forwarded based on PIT entries. The strong coupling of Interest and data flow requires that the MAC address assigned to the Interest face is in line with the data face. However, usually there is a time gap between sending Interest and forwarding corresponding data. A unicast MAC address that is valid during Interest submission might be outdated when data is forwarded. On the other hand, this case reduces radio transmissions and CPU processing to a minimum and fully incorporates MAC layer retransmission handling.

**Case 4: Interest Unicast, Data Broadcast.** The last case provides very limited redundancy. Data packets will be processed by the NDN stack of all nodes of the broadcast range. However, as Interest has been delivered via MAC unicast, only one node in the broadcast domain created a PIT entry. All other nodes will thus drop the data packet at network layer.

**Discussion.** Case 1 promises path and data redundancy but comes to the cost of excessive resource consumption which may be harmful, especially in IoT networks. Case 2 optimizes data transport via unicast but keeps forwarding redundancy and superfluity of a routing protocol. Case 3 fully optimizes resource overhead and transmission robustness which is promising for battery driven, constrained nodes. However, this approach requires a reliable routing mechanism since it minimizes path redundancy as well caching capabilities. Case 4 brings little benefit to NDN, as redundant data is not utilized.

### 4.2.2 The Case for Link Layer Assistance

**Experimental Exploration.** In this initial experiment, we want to check back on the effect of a reliable unicast link layer by counting incomplete Interest-data handshakes. For this, we select three nodes within radio range from the *Lille* site of the FIT IoT-Lab testbed (s. Section 4.3). One consumer node requests 1000 content items from one producer node at a rate of two Interests per second (without Interest retransmissions), installing different MAC layer mappings. The third node generates side traffic on the same radio channel, sending packets of 50 Bytes within random intervals between 3-10 ms.

**Results.** Table 4.1 presents unsatisfied Interests at the consumer as an indicator of packet loss (Interest or data). Strikingly, we see that broadcasting Interests increases the error rate by about one order of magnitude (Case 2 versus Case 3). Broadcasting data after Interest unicasts appear more robust, which we account to an implicit link layer coordination. In the presence of a periodic radio interferer, Interests are retransmitted on the MAC layer until the interferer paused and the transfer succeeded. Data in this single hop scenario follows immediately and

Table 4.1: Unsatisfied Interests with different face to MAC address mappings under presence of link layer interference.

| Interest \ Data | Broadcast | Unicast |
|---|---|---|
| Broadcast | 12.1 % (*Case 1*) | 10.6 % (*Case 2*) |
| Unicast | 3.3 % (*Case 4*) | 1.9 % (*Case 3*) |

thus takes advantage of the same pause (Case 2 versus Case 4). Broadcasting Interest and data (Case 1) combines these two sources of errors from Case 2 and Case 4. We conclude that NDN can substantially benefit from utilizing the support of MAC layer robustness.

## 4.3 Evaluation

The objective of our experiments is the measurement of basic effects through different MAC layer mappings in a common IoT environment. Thereby, we make use of standard software solutions and typical IoT hardware including low power radio transmission technologies. We will focus on Case 2 and Case 3 (see Section 4.2) because CCN-lite does not support data transmissions via broadcast in the current version.

### 4.3.1 Experimental Setup

All experiments are conducted in the FIT IoT-LAB testbed [5] to reflect common IoT properties. The testbed consists of several hundreds of class 2 devices equipped with an ARM Cortex-M3 MCU, 64 kB of RAM and 512 kB of ROM, and an IEEE 802.15.4 radio (*i.e.,* Atmel AT86RF231 [29]). The radio card provides basic MAC layer functions implemented in hardware, such as ACK handling, retransmissions, and CSMA/CA. For power measurements, we parameterize the consumption monitoring tool of the testbed with a conversion time of 332 $\mu$s and averaging over 64 samples. The software platform is based on RIOT [32] and the CCN-lite network stack [372], which we include as a third party library in RIOT. The integration of CCN-lite into RIOT and its default components are visualized in Figure 4.1.

We use default configuration parameters in RIOT and CCN-lite where possible and not mentioned otherwise. In detail, we deploy RIOT release 2017.01 and CCN-lite master with latest updates from May 10, 2017. For our measurements, we configure CCN-lite with a maximum of three Interest retransmissions and 12 seconds Interest timeout. MAC configurations of the radio devices enable IEEE 802.15.4 ACK requests for unicast traffic with a maximum of four retransmissions, and CSMA with a maximum number of four retries, introducing random delays after denied channel access (see [29] for further default values).

The subsequent experiments include single-hop and multi-hop scenarios, which we describe in

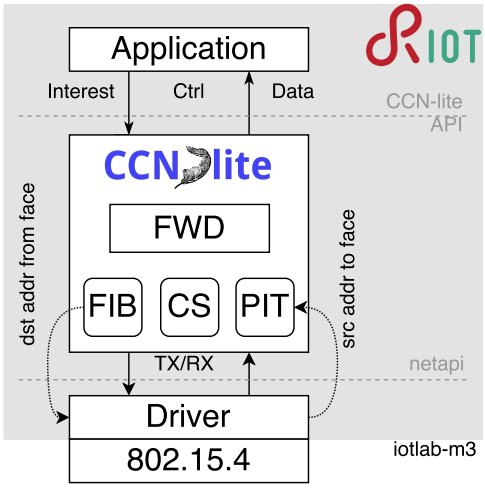



Figure 4.1: System environment: Integration of RIOT and CCN-lite to implement dynamic broadcast and unicast faces in NDN.

more detail next to the analysis of our experiments. Our results represent averages over multiple runs with the same parameter settings.

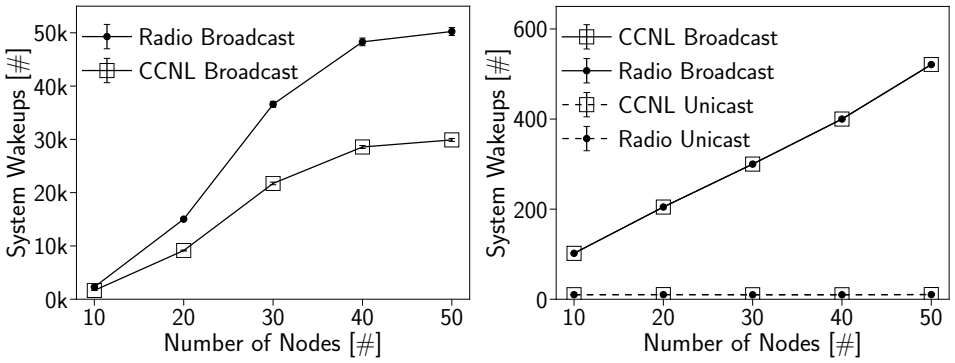
### 4.3.2 Single-hop Scenario

#### 4.3.2.1 Configuration

We deploy our single-hop measurements at the *Lille* site of the FIT IoT-LAB testbed because all nodes are located in the same broadcast range. We randomly select a single consumer node and a varying number of producer nodes for different measurement runs. Each producer is equipped with a different number of unique and static content items. In all subsequent scenarios, the consumer requests existing content items randomly. We measure the number of system wakeups and the CPU load of both the CCN-lite software stack and the radio device driver.

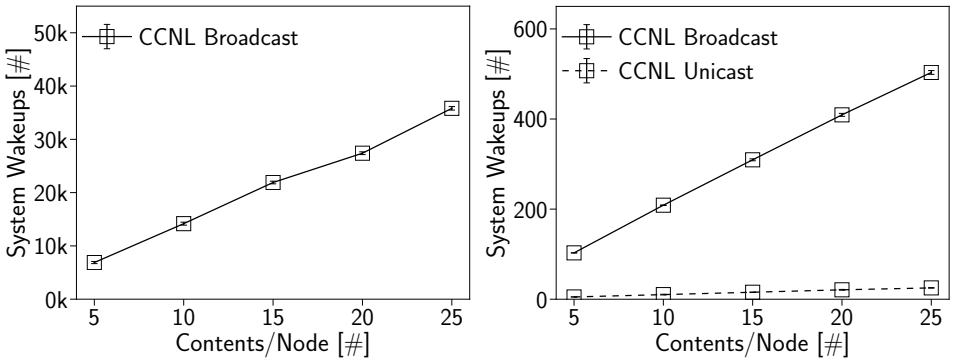
To implement single-hop data exchange on the data link layer and the network layer, all nodes need to be in physical reachability and consumers need to have routing entries that reach the producers directly. To consider common scenarios, we analyze three basic configurations. (i) On *all nodes*, we install a common prefix route that covers all content names, and the corresponding face refers to the broadcast address. Note, in this case, a unicast MAC address conflicts with reachability of arbitrary content items via a single hop as content is requested from multiple producers. In the remaining configurations, we install dedicated FIB entries *only on the selected consumer*, which refer (ii) either to unicast addresses of the producers or (iii) to the broadcast address.

Furthermore, to analyze different network sizes and load, we vary the number of producer nodes, or the number of content items per producer in a predefined network size.




(a) 10 contents items per node with prefix routes on all nodes.

(b) 10 contents items per node with prefix routes only on consumer.




(c) Network of 20 nodes with prefix routes on all nodes.

(d) Network of 20 nodes with prefix routes only on consumer.

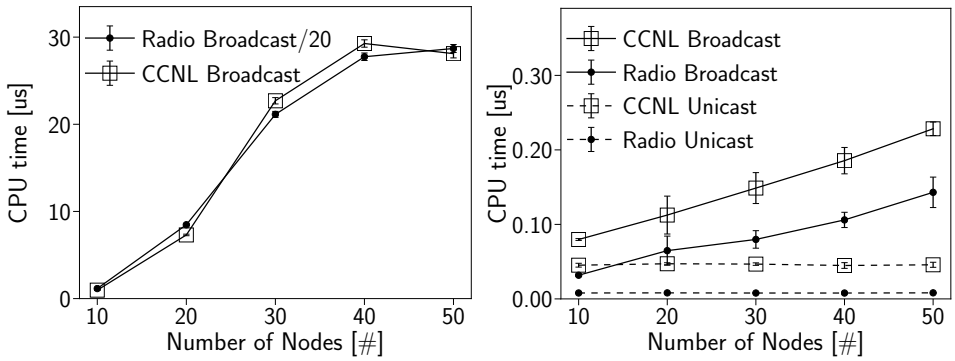
Figure 4.2: Number of system wakeups for varying network setups.

*Variable network size.* The number of content items per node is fixed, but we increase the number of producers in different parameter settings. We implement a fixed average content request rate per producer, *i.e.,* the number of Interests sent by the consumer increases linearly with the number of nodes in the network.

*Variable number of contents items per node.* The number of nodes is set to 20 and the number of content items per node is increased over different measurement runs. We apply a constant content request rate at the consumer.

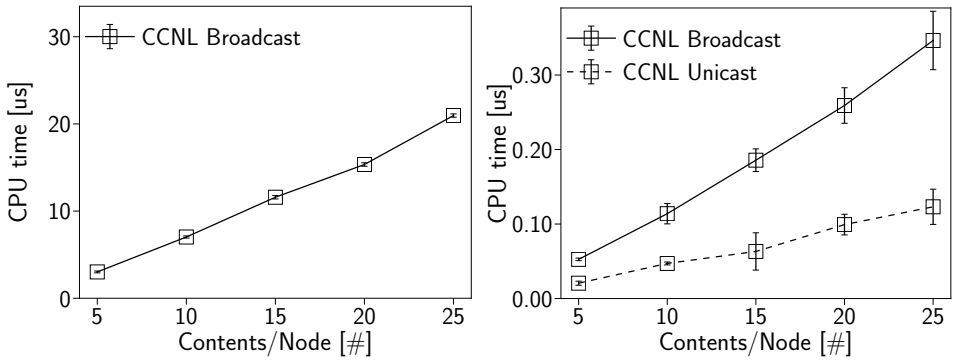
#### 4.3.2.2 Results

Figure 4.2 shows the number of system wakeups per producer for the single hop scenario, with variable network sizes and a fixed number of content items per node (see Figures 4.2a and 4.2b), as well as with a fixed network size and variable number of content items per node (see Figures 4.2c and 4.2d). Figures 4.2a and 4.2c represent the setup where Interests are sent to the broadcast MAC address and all nodes have routing entries for all content names.




(a) 10 contents items per node with prefix routes on all nodes.

(b) 10 contents items per node with prefix routes only on consumer.




(c) Network of 20 nodes with prefix routes on all nodes.

(d) Network of 20 nodes with prefix routes only on consumer.

Figure 4.3: Absolute CPU usage for varying network setups.

Figures 4.2b and 4.2d represent the setup where only the consumer node has FIB entries, which maps faces either to the unicast address of each content producer or to the broadcast domain. Correspondingly, Figure 4.3 represents statistics of the CPU usage we measured.

In terms of energy and processing overhead, it is clearly visible that faces with unicast MAC addresses outperform broadcast faces. While the number of system wakeups is constant for varying network sizes, it only increases in direct relation to the number of provided content items of one node with unicast mapping. Broadcast overhead increases linearly with the number of nodes *and* the number of contents per node, thus it directly correlates with the total number of requested content items in the whole (single-hop) network. This increases resource consumption.

To summarize, unicast faces can improve the lifetime of battery driven IoT devices by keeping CPU-wakeups and processing overhead at a minimum, and increase stability by benefiting from built-in MAC layer mechanisms for unicast traffic, such as ACK handling and retransmission. On the other hand, it requires a maintenance mechanism for the assignment of MAC addresses

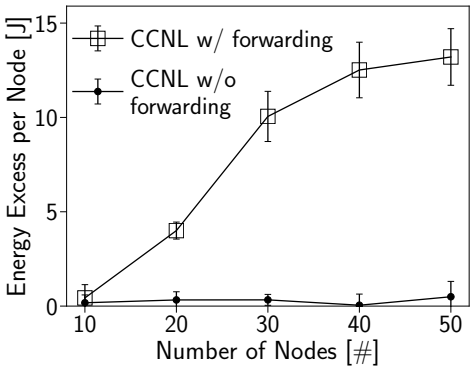


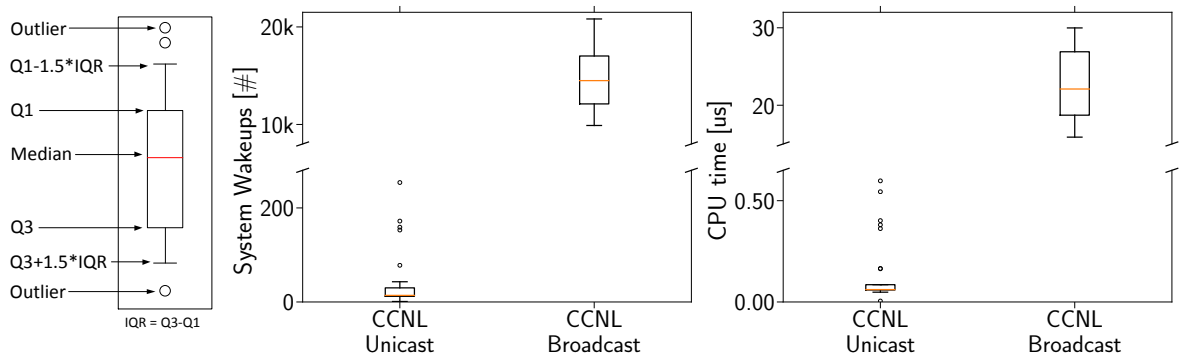

Figure 4.4: Average energy excess per producer: Broadcast with and without common prefix routes vs. unicast.

to faces and reduces redundancy by omitting built-in content replica-mechanisms as well as alternative data paths.

Deploying common prefix routes to broadcast faces on all nodes reduces the overall performance of the network even more, as each node in the broadcast domain does not only wake up during incoming Interests, but also forwards Interest packets that will not be satisfied, as well as data packets that might be received as a consequence of the forwarding mechanism. This leads to an excessive number of wakeups of all nodes in the domain as well as additional data transmissions. The overhead in this broadcast scenario is several orders of magnitudes higher than that of unicast.

To better understand the overhead introduced by broadcast Interest forwarding when all nodes store FIB entries, Figures 4.2a and 4.2b present wakeups, which are separately shown for the radio device and the CCN-lite (CCNL) network stack. In a setup consisting of a single application, single network stack, and a single network interface, both measurements should be roughly equal as the link layer forwards each broadcast packet up to the network stack and vice versa. This holds only in case of a single forwarder (see Figure 4.2b). We detect a much higher number of wakeups by the device driver when all nodes store routes. The impact on CPU times is worse by a *factor of 20*, as depicted in Figure 4.3a. We assume radio channel saturation causing this increased resource consumption. To further back these observations, we also measured (on the same network scale) (i) the rate of unsatisfied Interests (0 – 50 %), (ii) radio statistics from which we compute the rate of unsuccessfully transmitted packets due to failing CSMA/CA channel access (0,39 – 0,56), and (iii) the average number of network layer retransmissions (2 – 9 %). All these observables indicate a negative impact on network utilization while broadcasting.

Figure 4.4 displays the energy per node additionally consumed when Interests are broadcasted. We show the energy excess of *Interest broadcast, data unicast* with and without Interest forwarding over an *Interest unicast, data unicast* mapping for varying network sizes. By no surprise,




(b) System wakeups (c) CPU usage

Figure 4.5: Network of 30 nodes w/ 20 producers and 10 contents items per producer.

the graphs resemble Figures 4.2a and 4.2b, and the additional consumption with forwarding exceeds single-hop broadcast by orders of magnitudes, for increasing (single-hop) networks.

### 4.3.3 Multi-hop Scenario

#### 4.3.3.1 Configuration

We conduct our multi-hop measurements in *Grenoble*. This site of the testbed provides placement of nodes such that nodes do not form a single broadcast domain. However, fluctuating properties of the wireless media (*e.g.,* reflections) may lead to changing topologies from multi-hop to single-hop. To ensure a minimal multi-hop connectivity, we introduce a monitoring phase before our experiments start which is based on the mechanism proposed in [160]. During this phase, we identify a set of nodes that inter-connect over multiple hops. The resulting topology consists of 30 nodes where one node acts as consumer, 20 nodes act as producers with a distance of three hops towards the producer, and other nodes serve as intermediate nodes on the path.

Similar to the single-hop scenario, leaf nodes of the resulting topology are equipped with unique content items that are requested by a single consumer in randomized order. Referring to the single-hop experiments, we compare two mapping schemes from content names to faces at the consumer node and subsequent intermediate nodes: (i) a direct assignment of the next-hop MAC address to the corresponding face on the path to the producer, and (ii) a common prefix route where the corresponding face is mapped to a broadcast MAC address.

Even though the same set of nodes is used for (i) and (ii) we cannot guarantee that the same topology appears within the broadcast scenario, as discussed earlier. We do not consider this as a drawback but rather as an advantage, reflecting real-world properties.

#### 4.3.3.2 Results

In Figure 4.5, we show the impact of broadcast and unicast faces in a multi-hop network in terms system wakeups and CPU times for a fixed size network and a predefined number of content items per producer. We find similar effects compared to the single-hop scenario, where resource costs for the broadcast mapping (with common prefix routes applied on all nodes) are orders of magnitudes higher than for the unicast mapping. The medians of both wakeups and CPU times correspond to our single-hop measurements but larger errors and outliers are visible. The reason for the outlier is rooted in intermediate nodes. These nodes only forward Interest and data packets on the path between producer and consumer. In our measurements, we observed that single links which were stable during the monitoring phase, exhibit asymmetric link behavior later. That led to packet loss of approximately 10 % in the unicast setup, whereas the broadcast approach delivered 100 % of the requested content items due redundant paths. The resource improvement of name to unicast address mapping as well as the additional MAC layer features such retransmission handling come at the cost of a route maintenance mechanism that is needed to provide fresh and stable links. Analyzing this in more detail, should be part of future work.

## 4.4 Conclusions

In this chapter, we discussed current solutions to implement interaction between the NDN network layer and the underlying data link layer. In contrast to the IP network stack, which maps IP addresses to unicast MAC addresses to prevent arbitrary broadcast, there is no such mechanism by default in NDN. Without sacrificing the principle concepts of NDN, we argue that link layer broadcast should be reduced in specific deployment scenarios (*e.g.,* IoT), as it conflicts with limited hardware resources in terms of processing, memory, and energy. We reviewed the current solution space and contributed a first set of experiments in a real testbed. We linked NDN faces to unicast or broadcast MAC addresses and quantified the resource overhead and the advantage of using link layer functions (*e.g.,* retransmission handling).

The position of this chapter is threefold. First, a name to link layer mapping is needed and still an open research question. Second, our community should find a solution that does not affect the core of current link layer implementations, and benefits from built-in link layer functions. Third, one promising solution, the dynamic creation of NDN faces, has been mostly ignored and deserves more detailed study. In this chapter, we contributed a first set of experiments in a real testbed and related analysis.

# Chapter 5

# Decentralized MAC and Network Layer for LoRa


## Abstract

This chapter presents LoRa-ICN, a comprehensive IoT networking system based on a common long-range communication layer (LoRa) combined with Information-centric Networking (ICN) principles. We have replaced the LoRaWAN MAC layer with an IEEE 802.15.4 Deterministic and Synchronous Multi-Channel Extension (DSME). This multifaceted MAC layer allows for different mappings of ICN message semantics, which we explore to enable new LoRa scenarios.

We designed LoRa-ICN from the ground-up to improve reliability and to reduce dependency on centralized components in LoRa IoT scenarios. We have implemented a feature-complete prototype in a common network simulator to validate our approach. Our results show design trade-offs of different mapping alternatives in terms of robustness and efficiency.


## 5.1 Background and Challenges

**LoRa PHY: Long-range but very low data rates.** The LoRa PHY layer defines a chirp spread spectrum modulation which enables a long transmission range (theoretical 2–14 km) using minimal energy. Spreading factor (SF), code rate, and bandwidth can be configured and directly affect the time on air and data rate. As an example, a 50 Bytes frame has an on-air time of 2.3 seconds using SF12, code rate 4/5, 125 kHz bandwidth which leads to a PHY bit rate of 250 bit/s. Varying center-frequencies in the sub-GHz ISM band constrain the duty cycle to 0.1–10 % and further limit the effective throughput. As a consequence, the maximum effective bit rate of the physical layer can be as low as 0.25 bit/s.

While the LoRa PHY provides attractive features, it clearly imposes significant constraints with respect to worst-case latency and throughput, regardless of higher layer protocols such as the MAC layer. It is important to note that LoRa networks are therefore not comparable to

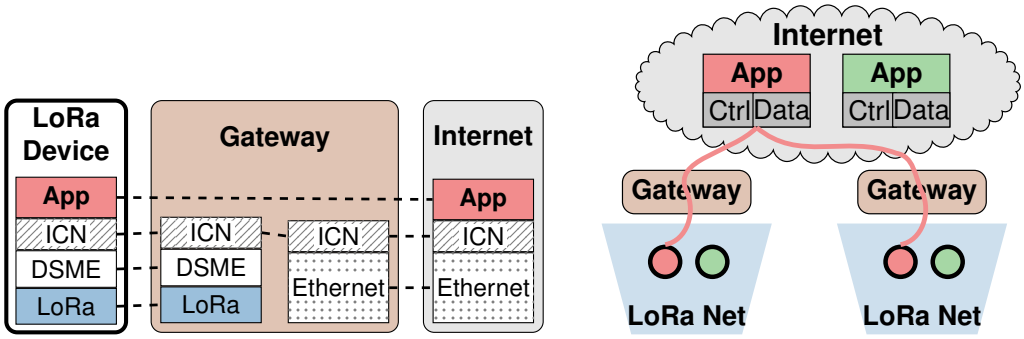
IEEE 802.11—instead they provide properties that incur significant challenges to higher-layer protocol design with respect to delay tolerance.

**LoRaWAN MAC Layer: Limited communication models.** The LoRaWAN MAC layer defines three operation modes: classes A–C. In class A, constrained Nodes send uplink using the ALOHA medium access protocol and can receive downlink traffic within two subsequent slots. This approach has three limitations: (*i*) ALOHA is susceptible to collisions. (*ii*) Downlink traffic is fairly limited and cannot be initiated by a Gateway. (*iii*) It prevents broadcast traffic. Class C works similarly to class A but leaves the radio always on, which enables Gateway initiated (multicast) downlink traffic but comes at high energy cost on constrained Nodes. Class B adds periodic slots which allows for "predictable" downlinks at medium energy consumption. Gateways send beacons every 128 seconds (time-synchronized by GPS). Consequently, hardware requirements are not compatible with current deployments that mostly serve class A. Beacons of neighbored Gateways can collide, for the absence of beacon synchronization. The Network Server arranges MAC schedules, however, downlink slots can overlap or be suspended. Scalability issues of class B have been analyzed in [345]. Furthermore, uplink traffic still uses ALOHA in class B mode, which interferes with downlink traffic, so that communication is still best effort.

**LoRaWAN Network Architecture: Gateway and server centric.** In the LoRaWAN architecture, radio networks are connected by Gateways to a Network Server that provides over-the-air activation, message deduplication, message routing, adaptive rate control at end devices, and acknowledging messages. Gateways are merely relays that implement timing-relevant aspects of the MAC protocol such as sending beacons. In the upstream direction, Gateways forward (tunnel) frames to the Network Server over the Internet. In the downstream direction, the Network Server sends LoRaWAN messages to LoRa Nodes, which includes Gateways selection.

The LoRaWAN specification does not mandate particular deployment options, and Network Servers could in theory be co-located to Gateways. In practice, *e.g.,* in public networks, such as TTN [75], Network Servers are operated in the Internet, and Gateways are peripherals of the Network Server, *i.e.,* they cannot operate without it. While this design decision is practical with respect to ease Gateway operation, it leads to a centralized architecture around the Network Server and additional servers such as Application Servers that provide interfacing to business logic, interconnecting local LoRa networks, and data sharing.

**DSME: Reliable and ICN-friendly MAC layer.** The 802.15.4 DSME MAC (see [174] for further details) enables new LoRa scenarios. A coordinator starts network formation and emits beacons in a pre-defined beacon interval (including beacon collision resolution mechanisms), to initiate a synchronized multi-superframe structure. A superframe in the multi-superframe consists of: Beacon period (BP), contention-access period (CAP), and contention-free period (CFP); the latter of which provides seven guaranteed time slots (GTS), multiplexed across 16 radio channels. Data is transmitted during the CAP, a pre-allocated slot in the CFP, or in an "overloaded" beacon. Battery-driven Nodes mostly sleep (*e.g.,* during the CAP), which




(a) Layer configuration. (b) Network example.

Figure 5.1: LoRa-ICN stacks and networks.

makes them unavailable for the coordinator. DSME provides *Indirect Transmission*, in which a coordinator indicates pending transactions with a beacon. This triggers the constrained Node to stay awake after the BP.

## 5.2 Design Goals

In LoRa-ICN, we are re-imagining the system architecture of long-range IoT communications, aiming to overcome the challenges described in the previous section. In our deployment scenarios, an application consists of the control and data consumer applications in the Internet and a set of LoRa-ICN Nodes (ICN producers and consumers), potentially distributed over multiple individual LoRa radio networks. Each LoRa network is served by one Gateway. The controlling and consuming applications can access all their associated wireless Nodes directly over the Internet, without mediation through application layer Gateways, see Figure 5.1.

Our design goals are (*i*) not requiring changes to the rest of the ICN network, (*ii*) providing a complete set of interaction patterns such as data transmission and Node control, (*iii*) leveraging the LoRa PHY capabilities optimally. To implement a fully distributed system model, LoRa-ICN Gateways operate as layer 3 routers instead of just bridges as in LoRaWAN. Key functions that are typically implemented by LoRaWAN Network Servers (*e.g.,* routing) are performed by Gateways.

In order to leverage the LoRa PHY capabilities and to support the rich ICN interaction patterns, we replace the LoRaWAN MAC layer with the significantly more powerful IEEE 802.15.4 DSME MAC layer [174] that provides better reliability (important for ICN Interests) and reduced latency (important for ICN consumer-publisher communication). In the following, we discuss the most relevant operations that support our deployment scenarios.

**Node Registration** refers to a Node registering its prefixes with the local Gateway (acting

as ICN forwarders), which will install FIB entries and announce the prefixes outside the LoRa network so that Nodes do not need to participate in routing.

**Data Provisioning by Nodes** on LoRa Nodes includes asynchronously produced sensor data as well as requested data transmission. ICN is a receiver-driven system, so we distinguish two main variants: (*i*) *ICN-idiomatic Interest/Data.* (*ii*) *Push Data from Nodes to Gateways.* While unsolicited push is not an ICN-idiomatic communication pattern, it is still a useful capability in a resource-constrained environment because IoT Nodes may need to save energy and produce data only occasionally.

**Node Control** refers to Nodes being reliably controlled by peers in the Internet, *e.g.,* for sensor control and configuration. We use a basic Interest-triggered interaction, and Data as ACK.

**Data Retrieval by Nodes** is natively enabled as LoRa-ICN Nodes are regular ICN Nodes and may also send Interests to other ICN Nodes hosted by the same LoRa network, other LoRa networks, or any other ICN network.

**Downstream Multicast** enables large-scale data distribution as needed for example in firmware updates. ICN features multicast via Node-generated Interests and broadcast Data messages from the Gateway.

## 5.3 ICN over LoRa

The LoRa PHY exhibits long on-air times (seconds) for transmit long- range (kilometers) at minimum energy consumption (micro-joules) and underlies rigorous duty-cycle restrictions. In order to achieve a robust system design, we proceed in two steps. First, we utilize a proven LoRa PHY configuration to leverage the DSME MAC. Next, we derive a viable mapping of ICN to DSME/LoRa for sending ICN Interest and Data messages from and to LoRa Nodes.

### 5.3.1 Mapping DSME to LoRa

We apply the PHY mapping presented by Alamos *et al.* [10] to utilize LoRa below DSME. This includes a spreading factor of 7, a bandwidth of 125 kHz, and a code rate of 4/5. The beacon interval is 125.82 s to align with LoRaWAN class B beacons (128 s). The contention-free period (CFP) defines 16 channels with a 1 % duty-cycle restriction. Beacons and CAP use a common channel of 10 % duty cycle. During CAP, Nodes perform CSMA-CA and incorporate channel activity detection (CAD) of common LoRa devices.

The CFP channels are designed to carry data of high reliability and limited latency demands. The time division of DSME, though, requires a packet queue and hence affects transmission speed. Traffic load determines queue occupation. We want to estimate the average waiting time (*i.e.,* time in queue) for a packet that should be transmitted reliably during the CFP.

Little's law [241] $W = L/\lambda$ approximates the average waiting time $W$, using the average number of items $L$ (*i.e.,* queued packets) at a given average arrival rate $\lambda$ (packet rate). We

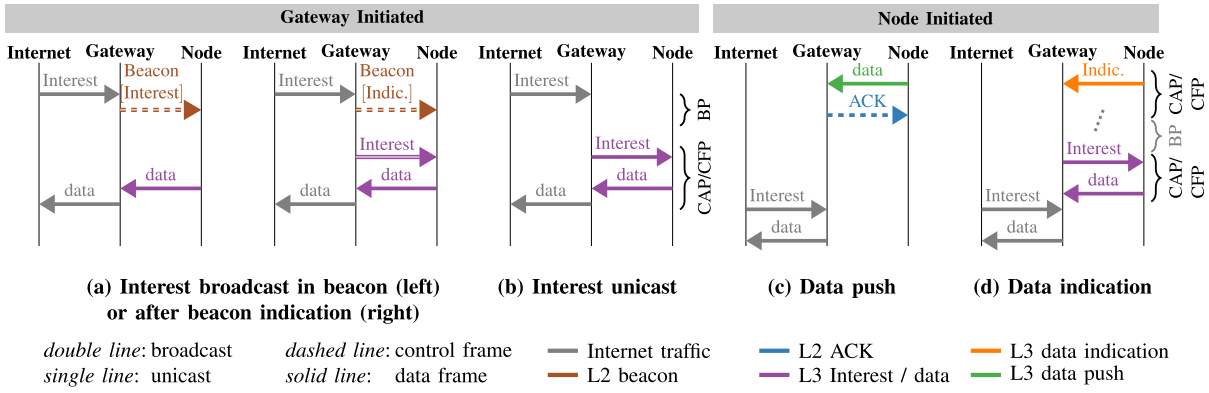



Figure 5.2: Mapping schemes of Interest/Data and ICN extension packets to DSME frames. Data flow is from Node to Gateway, either initiated by the Gateway ((a)-(b)) or by the Node ((c)-(d)).

assume (*i*) independent, exponentially distributed inter arrival times of packets with an expectation rate of $\lambda$. (*ii*) Nodes allocate only one transmission slot in CFP and (without loss of generality) (*iii*) the MAC queue has infinite capacity.

Let $L(t_n)$ be the number of queued packets after the transmission time $t_n$ of the $n$-th multi-superframe. We note that $(L(t_n))_n$ is an ergodic Markov process (positive recurrent and aperiodic), for which the limiting distribution $\pi_i$ exists (with $i = 0, \ldots, \infty$ the number queued packets). This stationary eigenvector can be calculated numerically as a fixed point of a (clipped) high-dimensional transition matrix and yields the stationary mean occupancy $L(t_\infty)$ prior to starting the new superframe. The actual queue that an arriving packet faces holds also packets which arrived during the current multi-superframe (of duration $T$), *i.e.,*

$$L = L(t_\infty) + \frac{\lambda \cdot T}{2} \tag{5.1}$$

As an example, we choose a relaxed packet arrival rate $\lambda = 1/120\,\mathrm{s}$ and compose our multi-frame structure of 4 superframes, *i.e.,* $T = 32.46\dot{s}$. This results in an average number of $L \approx 0.18$ queued packets and an average waiting time of $W \approx 21.32\,\mathrm{s}$ respectively. This scenario is compatible to the downlink in a common class B configuration, which exhibits an average waiting time of 44 s but suffers from 26% loss [101]. In contrast, loss is very unlikely in our CFP time division period.

### 5.3.2 A MAC for ICN using a LoRa-Proxy

Figure 5.2 presents options to handle Interest and Data packets between a high throughput network (*e.g.,* the Internet) and a DSME/LoRa network. The left part (Figure 5.2(a)-(b)) shows Gateway-initiated request-response communication, resembling native ICN primitives.

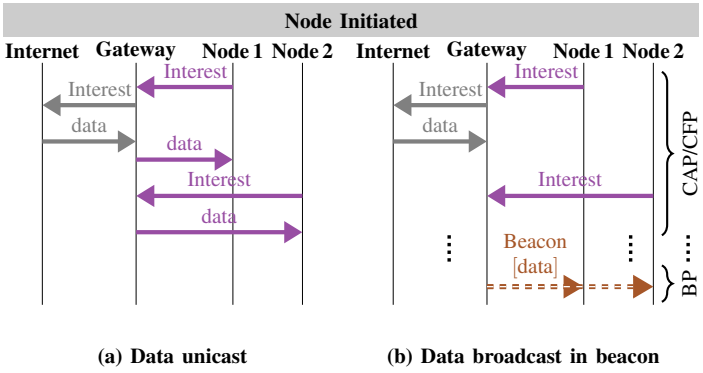



Figure 5.3: Mapping schemes of Interest/Data and ICN extensions packets to DSME frames. Data flow is from Gateway to Node.

In addition, we present protocol extensions for Nodes to initiate traffic (Figure 5.2(c)-(d)) to Gateways that act as proxies for constrained Nodes during sleep time.

**LoRa→Internet. Broadcast.** Beacons are regularly broadcast by Gateways and can carry Interests without message overhead. This maximizes sleep cycles, but the limited beacon intervals reduce throughput. Using beacons to transfer Interests provides two options (see Fig. 5.2(a)). (*i*) Beacons carry payloads up to a frame size of 127 Byte minus metadata, *i.e.,* ≈ 100 Bytes/frame. Using ICNLoWPAN encoding [148] this is sufficient to aggregate 4–6 Interest packets. (*ii*) Gateways utilize indirect transmission (see Section 5.1) to broadcast an Interest, which involves the indication of a pending transaction within the beacon, and subsequent Interest broadcast during the CAP.

**LoRa→Internet. Unicast.** Interest and Data messages can be sent via unicast within the CAP or CFP (see Fig. 5.2(b)). Sending Interests in best effort CAP frames enables requests at higher rates than in beacons. Nevertheless, this prevents Nodes from sleeping during the CAP. Note that individual Interests increase the number of downlink packets from the Gateway; for growing wireless networks this conflicts with duty cycle restrictions at the Gateway. Using the CAP for Data instead is less critical since transmissions are initiated in the low-power Node. The CFP provides exclusive resource access but adds the overhead of a preceding cell negotiation, and is limited within the superframe structure. Hence, a full CFP Interest-Data exchange requires two cell allocations per Gateway-Node pair.

**LoRa→Internet. Data indication.** Nodes can offload the Gateway by removing the need for polling. This is done indicating names for subsequent Interests from the Gateway (see Fig. 5.2(d)). The indication packet, however, adds wireless traffic. Since Nodes do not emit beacons, indication packets utilize unicast traffic in the CAP or CFP. Interest and Data packets follow as outlined in Figs. 5.2(a) or (b). Hence, an indication and the following Interest broadcast must

wait for the next beacon period. Instead, Interest unicast in CAP or CFP keeps the latency for producer-initiated traffic minimal.

**LoRa→Internet. Local Data push.** A link-local Data push from producers to the Gateway reduces radio access and maximizes device sleep times, similar to LoRaWAN class A deployments (see Fig. 5.2(c)). An optional link layer ACK with retransmissions from the Node increases reliability in the CAP; the exclusive CFP slots can omit the ACK.

**Internet→LoRa. Unicast.** Nodes request data from a Gateway by sending Interests within CAP or CFP frames (Figure 5.3 (a)). Data returns in a CFP cell pre-allocated for every consumer Node. This provides high reliability but becomes challenging in larger networks, since the amount of downlink cells is limited by the multi-superframe.

**Internet→LoRa. Broadcast.** Multiple Interests for the same content item arrive at the Gateway that responds with a Data broadcast message (see Figure 5.3(b)) using an overloaded beacon. This can help with observing duty cycle restrictions on the Gateway; however the data throughput is limited by the beacon interval.

## 5.4 Simulation Environment

We have developed a simulation environment for LoRa-ICN that is based on OMNeT++ and the INET framework [176]. We integrated ccnSim [71] for core ICN support and openDSME [186] for 802.15.4 DSME functionality. Our model uses FLoRa [349] and its wireless propagation model and PHY. Figure 5.4 depicts the simulation environment and our extensions.

**Data flows** orchestrate ICN Interest/Data exchanges and are adjusted to match IoT use cases as follows: We changed the built-in content popularity model from Mandelbrodt-Zipf to a uniform distribution and initiate one transmission for every content item. We also extended the ccnSim core implementation by two network layer primitives: (*i*) *Indication* (see Section 5.3.2) and (*ii*) *Push* to place a data item in the neighboring content store. In all scenarios, content rates follow a Poisson process.

**ICN-to-DSME** addresses three main challenges. (*i*) ccnSim lacks the concept of a link layer. Instead, ICN faces directly connect to I/Os of neighboring Nodes. We include a wireless transmission link. (*ii*) We add a face-to-MAC module that multiplexes ICN faces to a *Wireless-Interface* and uses MAC addresses for transmission. This module includes all logic for the ICN-to-DSME mapping (see Section 5.3.2). (*iii*) OMNeT++ messages are converted into INET packets that map to openDSME. This step includes tagging of packets and appends control instructions for the MAC layer.

**DSME-to-LoRa** integrates the LoRa PHY with the DSME MAC implementation. This component bases on related work, and we refer the reader to Alamos *et al.* [10]. We further disable dynamic slot allocation for the CFP to exclude negotiation overhead, but implement static scheduling and MAC configurations. In bidirectional communications, each Tx slot is

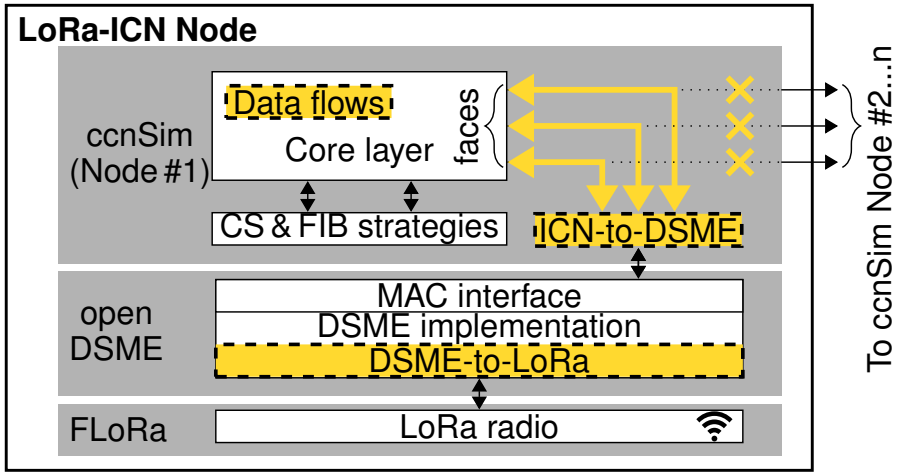



Figure 5.4: Simulation environment and our extensions.

followed by an Rx slot which halves the number of transactions in one multi-superframe. Every simulation Node is assigned zero, one, or two slots depending on the MAC mapping. This pattern repeats with a multi-superframe – with adjustable structure to simulate different network sizes.

## 5.5 Evaluation

We have implemented and tested different options for ICN-to-DSME mappings for the two major use cases *Data from Node to Gateway* and *Data from Gateway to Node*.

### 5.5.1 Data from Node to Gateway

**Motivation:** Table 5.1 presents an overview of the performance for ICN/DSME/LoRa in a network of 14 Nodes, when the Node is a producer, and data flows towards the Gateway. For **Gateway-initiated** traffic, **Interest broadcast** reflects a special case which is heavily limited by the beacon interval. With an aggregation of multiple Interests into one broadcast message, the Gateway is able to send $\approx 5$ Interests encoded in one broadcast packet, every $\approx 126$ s (beacon interval). Consequently, we can accommodate up to 5 Nodes responding with one Data message each in the same multi-superframe.

For the other mappings (see Section 5.3.2) in Table 5.1, each of the 14 Nodes produces a content item in one minute intervals (on average), and we disable Interest retransmissions to evaluate the plain ICN performance over LoRa. **Interest unicast** is separated into **Interest-CAP and CFP** variants (Interest-CAP prevents Nodes from sleep and is not feasible for battery powered devices). **Interest-CAP** provides short completion times of 8–12 s on average, due to frequent CAP intervals. **Interest-CFP** is slower by a factor of $\approx 1.5$. However, sending data in the CAP increases the probability for data loss, which is most notable when Interest and Data messages share the CAP. The maximum latency increases up to 56 s as a consequence of

Table 5.1: Performance overview of mapping schemes.

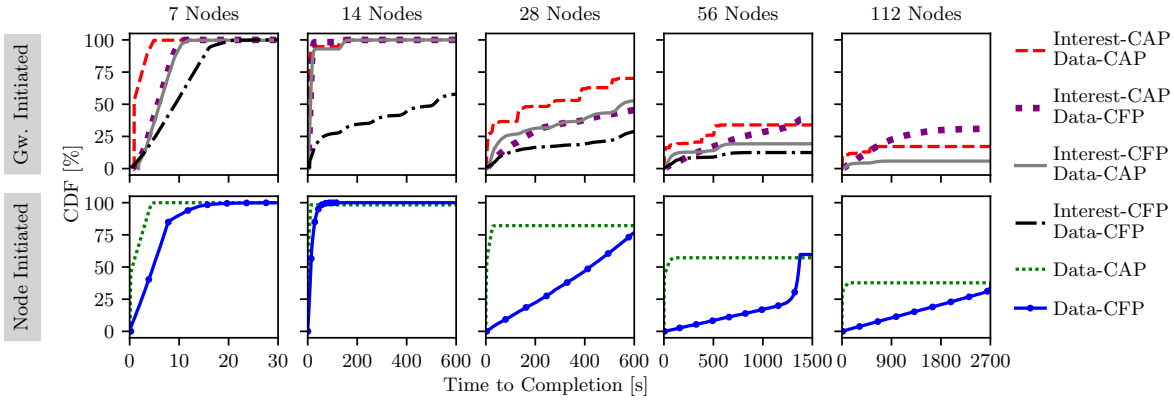
| | Mapping Scheme | Indication | Interest | Data | Avg. Latency [s] | Max. Latency [s] | Data Loss [%] |
|---|---|---|---|---|---|---|---|
| Gateway Initiated | Interest broadcast | | Beacon | CAP | (Not operable at this scale) | | |
| | | | Beacon | CFP | | | |
| | Interest unicast[1] | | CAP | CAP | 8.32 | 56.56 | 5.16 |
| | | | CAP | CFP | 11.67 | 26.87 | 0.05 |
| | Interest unicast | | CFP | CAP | 12.73 | 28.27 | 1.72 |
| | | | CFP | CFP | 17.01 | 47.24 | 0.00 |
| Node Initiated | Data indication | CAP | CFP | CAP | 19.25 | 75.17 | 3.13 |
| | | CAP | CFP | CFP | 14.61 | 71.61 | 1.51 |
| | | CFP | CFP | CAP | 71.62 | 299.22 | 2.6 |
| | | CFP | CFP | CFP | 46.49 | 241.96 | 0.89 |
| | Data push | | | CAP | 7.02 | 29.88 | 1.62 |
| | | | | CFP | 10.63 | 66.42 | 0.00 |

[1]Nodes must be turned on during CAP, which prevents low-power.

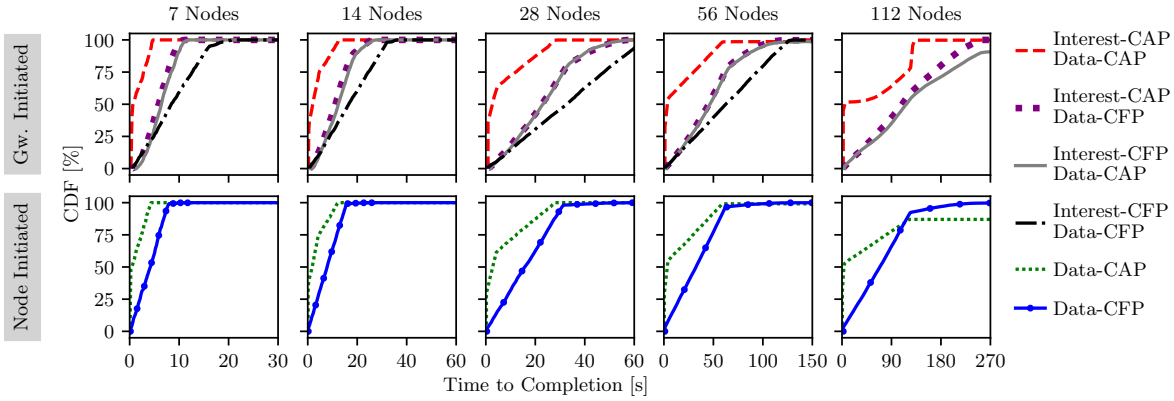
CSMA retries. Conversely, sending Data messages in the CFP improves reliability at moderate overhead for the maximum latencies, despite less frequently available GTS for sending.

For **Node-initiated** traffic, we compare **Data indication** (a dedicated indication messages is triggering an Interest by the consumer) and **Data push**. In the indication case, we only consider energy-efficient options in which the Gateway uses the CFP for Interests. Our results clearly show an overhead for the three-way handshake with Data indications. Maximum latencies increase to over 70 s using the CAP for indication, and to over 240 s using the CFP, even in this unstressed scenario. Hence, we ignore Data indication for the remainder of our evaluation. In contrast, **Data push** can obviously be completed in a single message transmission and reduces the average completion time to 7–11 s, depending on the CAP/CFP mapping variant. **Sending Data in the CAP** is affected by collisions and CSMA retries. **Sending Data in the CFP** surprisingly reveals a maximum latency of over 60 s. We ascribe this to the randomized content creation interval that leads to occasional synchronized medium access of several Nodes and then consequently to MAC layer queue processing delays that last for multiple multi-superframes (here 2) with static slot assignment.

Figure 5.5 presents completion times for Data retrieval from LoRa Nodes in a high data rate scenario (30 s Data production intervals, 5.5a) and a more relaxed scenario (900 s Data produc-




(a) Content at 30 s interval per Node



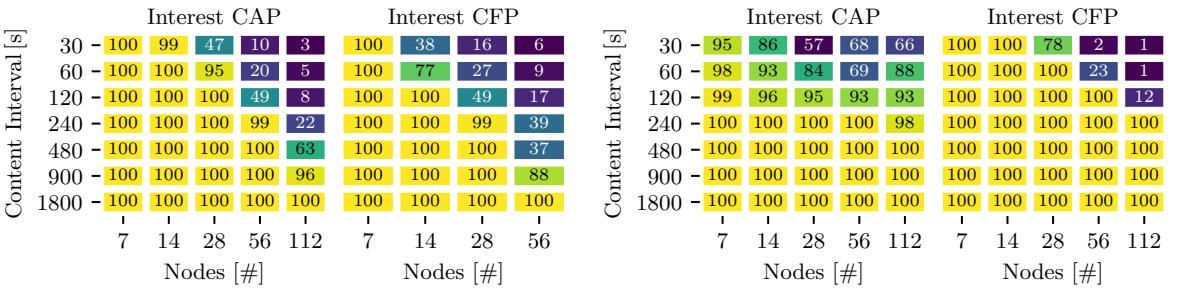

(b) Content at 900 s interval per Node

Figure 5.5: Time to content arrival with different mapping schemes for Interest/Data (with L3 retransmission) and Data push for varying network sizes.

tion intervals, 5.5b) including ICN retransmissions. Data losses result in infinite completion times, hence, the end value of each graph also reflects its success ratio.

**Performance at high data rates:** The **Gateway-initiated** requests finish in less than 20 s, in a small network of 7 Nodes. The performance degrades with increasing networks. In the 14 Nodes case, 90 % of the requests are satisfied in less than 30 s except for the fully CFP-based mapping. Missing Data triggers an Interest retransmission (step at $\approx$ 126 s) which reflects our retransmission timeout that aligns with the beacon interval. The Interest-CFP/Data-CFP mapping, however, finishes after 600 s with $\approx$ 50 % loss, which is the effect of MAC queue utilization that has a service rate at the order of one superframe ($\approx 32\,s$). Network sizes $\geq 28$ increase the completion time to the order of hundreds and thousands of seconds at high loss rates, which then becomes unusable for ICN communication.

Operating in a full CAP mapping exhibits the highest ratio of transactions successful at the first attempt, but it inhibits Nodes sleeping. Furthermore, 112 Node networks reduce




(a) Data unicast (w/ L3 retransmission)

(b) Data broadcast (w/o L3 retransmission)

Figure 5.6: Success rates [%] depending on network sizes and content invervals for different ICN mappings.

the delivery rate to 25 % due to collisions and denied channel access. Conversely, Interest-CAP/Data-CFP efficiently combines the "reactive" contention-based access for Interests with reliable contention-free media access for Data. Interest-CFP/Data-CFP with 112 Nodes is not possible with our current CFP scheduling approach (see Section 5.4).

**Node-initiated** Data push reveals faster completion and higher reliability in comparison to the request-based pattern. Networks $< 28$ Nodes show similar behavior for Data-CAP and -CFP transmissions. Conversely, networks $\geq 28$ show the effect of MAC over-utilization for Data-CFP. The multi-superframe period increases with the number of Nodes, hence, the average service rate of the MAC decreases. CAP reduction [174] can mitigate this effect and will be evaluated in future work. In contrast to Data-CFP, Data-CAP performs comparably smooth and transmits $\approx 50$ % of the messages within 30 s even under these stressful conditions.

**Performance at low rate:** In the relaxed scenario (Figure 5.5b), **Gateway-initiated** requests perform mostly reliable for all mappings and complete with 100 % success after less than 126 s in networks $< 112$ (note the change of the x-axis scale in comparison to Figure 5.5a). For 112 Node networks, Interest-CAP/Data-CAP clearly shows the effect of CSMA retries.

For **Node-initiated** traffic, Data-CFP is now on-par with Data-CAP and exhibits the best performance due to the contention-free media access. In the 112 Node network, the unidirectional push with Data-CAP faces collisions that cannot be compensated due to the absence of Interest retransmissions (in contrast to the Gateway-initiated case). Consequently, Data-CFP is the better option for large networks.

### 5.5.2 Data from Gateway to Node

**Data unicast:** Figure 5.6a depicts success rates for Nodes sending Interests to the Gateway with Interest-CAP and Interest-CFP mappings. Latencies are less crucial in this case since Nodes are aware of the constrained regime and sleep during long round trips. A retransmission will likely be answered by the Gateway if the requested Data has arrived in the meantime. Here,

we apply Data-CFP to enable maximum sleep times. Interests are sent in the CAP or CFP and experience similar challenges as described in Section 5.5.1. Again, fully CFP-based mappings are not available for 112 Node networks with the current scheduler.

Figure 5.6a clearly shows the network operation boundaries by a diagonal in the heatmap at 30 s/28 Nodes–240 s/112 Nodes for Interest-CAP, and 30 s/14 Nodes–120 s/240 Nodes for Interest-CFP. In relaxed scenarios, both options perform similarly well. Surprisingly, Interest-CAP outperforms the reliable CFP alternative for larger networks, despite its best-effort limitation. Hence, Interest scheduling in a GTS is more susceptible to losses than concurrent channel access, however, there is a crucial caveat that our measurements cannot exhibit: In a deployment with multiple Gateways (in reach), other stub networks share the CAP which increases the collision probability. In contrast, CFP slots follow a channel hopping scheme to avoid interfering Nodes. Furthermore, neighboring LoRa networks can assign the same GTS on orthogonal channels, which increases the overall throughput. We will focus on multi Gateway scenarios in future work.

**Data broadcast:** Figure 5.6b depicts success rates for Interests in CAP or CFP and subsequent Data broadcasts with indirect transmission, triggered by the beacon sent by the Gateway. Similarly to Interest broadcast (see Section 5.5.1), the maximum throughput of broadcast Data packets is limited by the beacon interval. In contrast, however, Data broadcast can satisfy many pending Interests that have been aggregated during the beacon period with a single downlink packet. Applying the ICNLoWPAN encoding, our approach concatenates up to six data items into one packet. This requires a fixed Interest window size for all Nodes within one beacon interval and homogeneous content requests during this period.

Figure 5.6b depicts that the best broadcast performance can be achieved without Interest retransmissions with request intervals at the order of 120 s or greater, which is in line with the beacon period. The impact of the network size is less significant in comparison to Figure 5.6a, which emphasizes the advantage of Data broadcast. Success rates for short request intervals $< 120$ s exhibit the advantage of Interest-CFP over CAP. Exclusive uplink resources are less susceptible to interference, whereas the shared media access during CAP suffers from limited media access and collisions.

Interest retransmissions worsen the success rate for networks $\geq 56$ Nodes in this scenario, for two reasons: (*i*) delayed request of "old" Data occupies the limited downlink resources of the Gateway, whereas neighbor Nodes simply have to drop duplicate Data. (*ii*) additional transmissions increase media access contention during CAP and stress the MAC queue during CFP.

## 5.6 LoRa-ICN Convergence Layer

Our simulation results revealed that for both upstream and downstream messages the CFP variants generally provide the best compromise between low-latency and overall throughput

across the wide range of scenarios we investigated. This leads us to the following approach for ICN Interest/Data exchanges. We include the operations described in Section 5.2.

Interests from consumers reach LoRa-ICN producers via the Gateway as per regular ICN forwarding. The Gateway forwards each Interest that matches a registration in a CFP slot and sets the expiration timeout to $10 \times n$ seconds, with $n$ set to the number of registered Nodes. The Gateway should perform Interest aggregation, *i.e.,* suppress duplicate Interests with the same name. Interests with unknown prefixes are NACKed. Nodes reply to these Interests with a Data (or a NACK) message in their assigned CFP slot. This message consumes the Interest on the Gateway as per regular forwarding behavior. Gateways cache the content objects in their content store. Depending on the LoRa network utilization, Interests may expire at the original consumer or at on-path forwarders. In such cases, consumers should re-issue the Interest, possibly increasing the Interest expiration time.

**Node Registration** is built on Interests that are transmitted from the Node to the Gateway and adopt NDN prefix registration [236]. Gateways propagate their registered prefixes to adjacent routers, unless scenarios demand otherwise. Registration state needs to be refreshed every 60 minutes. Similar mechanisms could be used to install per-Node filters to have more fine-granular control over Interests that are forwarded to the Node.

**Data Provisioning by Nodes** uses unsolicited push (Data messages) as the primary upstream Data communication primitive, thereby, utilizing a CFP slot. Gateways will accept unsolicited Data messages from Nodes that fall under the registered prefix and act as a custodian, *i.e.,* they will keep corresponding Data objects in their content store and satisfy matching Interests from the Internet. Gateways should store these objects for several minutes.

For Interests that cannot be satisfied from the content store, the Gateway performs normal forwarder operations, *i.e.,* it forwards the Interest to matching Nodes following the prefixes obtained from Node registrations.

**Node Control** actions are triggered by Interests from the Internet that are intended as a Remote Method Invocation (RMI). They use the same mechanism as other Interest from Internet consumers (see above). While this enables basic Node control, it has the usual problems of using Interest-Data for RMI as described in [211].

**Data Retrieval by Nodes** is implemented by Nodes sending Interests in their CFP slot, setting their local expiration time to 600 seconds. Gateways either consume or forward the Interest as per regular ICN forwarding. Corresponding Data objects are cached so that potential Interest retransmission can be satisfied by the Gateway.

**Downstream Multicast** can be used for synchronized downloads in a radio resource-efficient way. We assume that this would be triggered by a control command, possibly referring Nodes to a Manifest pointing to the actual Data objects. Possible optimization (*e.g.,* Gateway-controlled Data rates) will be studied in future work.

## 5.7 Related Work

**ICN and the IoT.** Our work is based on four observations of prior work. (*i*) the IoT benefits from ICN [33]. (*ii*) ICN should not ignore the MAC [193], to comply with constrained resources. (*iii*) to allow for periodic sleeping of devices without sacrificing performance, aligning ICN principles to lower layer frequency- and time division provides a unique opportunity. In contrast to prior work, which presented a design for ICN and 802.15.4 TSCH mode [174], we focus on LoRa and DSME.

**Analysis of 802.15.4-based Standards.** Comparing TSCH and DSME based on simulations is common [13]. Choudhury *et al.* [72] deploy DSME on constrained Nodes and found that TSCH obtains lower latency and higher throughput for small networks. DSME outperforms TSCH for higher duty cycles and an increasing number of Nodes, though.

Tree-based routing over DSME [218] has been proposed. Our topology choice is also supported by the IEEE 802.15 group which suggests long-range radios operating in star topologies.

**Analysis of LoRa and LoRaWAN.** Liando *et al.* [237] provide real-world measurements of LoRa and LoRaWAN. Saelens *et al.* [321] add listen-before-talk techniques to overcome band-specific duty cycle restrictions. Orfanidis *et al.* [296] find cross-technology interference between LoRa and 802.15.4 sub-GHz radios and propose an advanced CCA mechanism for mitigation. Mikhaylov *et al.* [264] reveal energy attack vectors in LoRaWAN, and Shiferaw *et al.* [345] present scalability issues with LoRaWAN class B. All those results indicate that LoRa and LoRaWAN suffer from scalability issues and are vulnerable to interference—which motivates our work.

**Alternative Protocols for LoRa.** Multi-hop routing in LoRa systems has been analyzed [80], including a replacement of the LoRaWAN MAC by LoRa and IPv6 to make use of RPL implementations for multi-hop networks [367]. Abrardo *et al.* [2] demonstrate a duty-cycling MAC layer to improve sleep time of constrained LoRa devices. Lee *et al.* [227] introduce LoRa mesh-networking that follows a request-response pattern and indicates performance benefits over producer-driven ALOHA. NDN was deployed on LoRa radios [243] which showed the need for a MAC layer. NDN over WiFi and LoRa [204] was proposed to connect 'isolated regions', however, nothing was mentioned about the LoRa MAC and how it prevents wireless interference and energy depletion.

For contention-based MAC, experiments with CSMA and CAD in LoRa-type networks indicate performance gains [305], without providing specific LoRa measurement results, though. Logical channel LoRa PHY configurations may assist frequency- and time division multiple access protocols [130]. Listen-before-talk in sub-GHz bands [233] performs better than ALOHA in LoRaWAN, and unconfirmed messages perform better in dense deployments.

Adaptations for time-slotted [418] LoRa have been presented in [315, 168] but consist of only three Nodes and limited traffic (12 packet/h). Alamos *et al.* [12] introduce DSME-LoRa and deploy 15 nodes. In this chapter, we close the gap by analysing reliability in larger networks.

## 5.8 Conclusions

The LoRa PHY is a radio layer that caters to many long-range, low-power communication scenarios. Unfortunately, the commonly used upper layer, LoRaWAN, is a vertically integrated communication system that cannot provide direct Internet connectivity and direct data sharing, but leads to centralized system architectures of limited scalability.

We introduced LoRa-ICN to overcome these limitations. We designed a new LoRa system from the ground up, leveraging the existing LoRa PHY but employing IEEE 802.15.4 DSME as a MAC layer and ICN as a network layer. To that end, we have defined a suitable DSME configuration, specified mappings of ICN protocol messages to DSME mechanisms, and proposed specific ICN extensions and Node requirements. Our DSME implementation provides the benefits of horizontal scalability, deterministic media access, and low-power operations. We could show in simulations for common network sizes that ICN messages gain reliability and reduce latency when mapped to DSME-CFP messages.

To support the current most relevant use case of IoT data transmission from constrained devices to the Internet well, we added a data custodian feature to Gateways. The result is a new LoRa system that supports direct end-to-end communication with LoRa Nodes and that can provide additional features such as downstream multicast natively. We claim that this highlights the versatility of ICN as an IoT network layer: By leveraging and minimally extending standard ICN caching, a LoRa Gateway can connect a delay-prone LoRa network to the Internet, without requiring any application awareness or protocol translation.

# Chapter 6

# Delay-tolerant Networking with ICN


## Abstract

Connecting long-range wireless networks to the Internet imposes challenges due to vastly longer round-trip-times (RTTs). In this chapter, we present an ICN protocol framework that enables robust and efficient delay-tolerant communication to edge networks. Our approach provides ICN-idiomatic communication between networks with vastly different RTTs. We applied this framework to LoRa, enabling end-to-end consumer-to-LoRa-producer interaction over an ICN-Internet and asynchronous data production in the LoRa edge. Instead of using LoRaWAN, we implemented an IEEE 802.15.4e DSME MAC layer on top of the LoRa PHY and ICN protocol mechanisms in RIOT OS. Executed on off-the-shelf IoT hardware, we provide a comparative evaluation for basic NDN-style ICN [413], RICE [211]-like pulling, and reflexive forwarding [295]. This is the first practical evaluation of ICN over LoRa using a reliable MAC. Our results show that periodic polling in NDN works inefficiently when facing long and differing RTTs. RICE reduces polling overhead and exploits gateway knowledge, without violating ICN principles. Reflexive forwarding reflects sporadic data generation naturally. Combined with a local data push, it operates efficiently and enables lifetimes of >1 year for battery powered LoRa-ICN nodes.


## 6.1 Background

In this section, we describe properties of the LoRa environment and the DSME MAC layer that our work is based on.

### 6.1.1 LoRa and LoRaWAN

LoRa defines a chirp spread spectrum modulation which enables a long transmission range (kilometers), low energy consumption (millijoules) at the cost of long on-air times. Duty cycle regulations further limit the effective throughput (bits per second). These features are still attractive for many IoT use cases. We operate on the EU 868 MHz band and configure a spreading factor 7, 125 kHz bandwidth, code rate 4/5, which results in a symbol time of 1.024 ms.

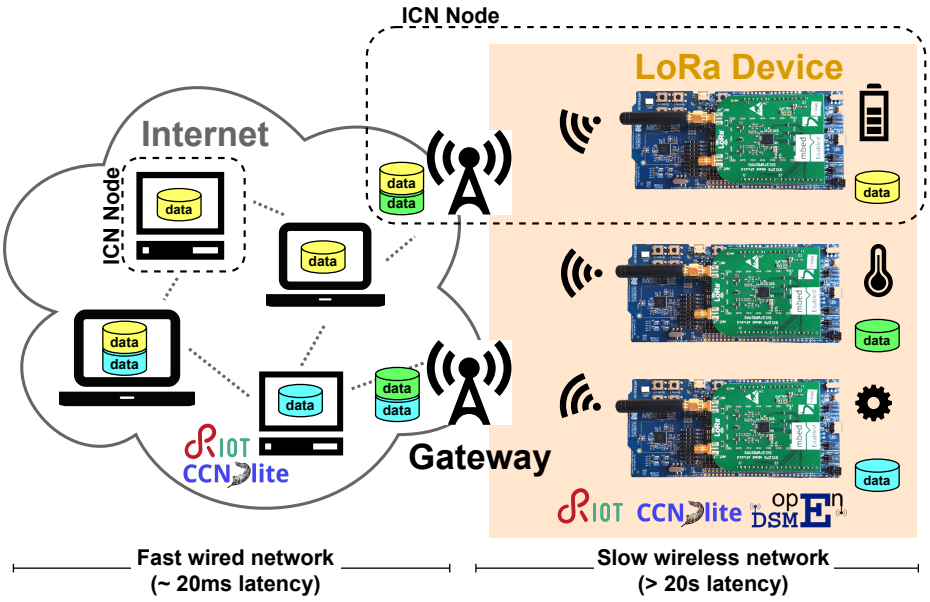



Figure 6.1: LoRa-ICN network and time domains.

LoRaWAN [245] is a popular system that operates on top of the LoRa PHY. It defines a vertically integrated, and centralized network architecture to integrate LoRa nodes to the IoT. So-called network- and application servers provide interfacing to the system. A network server interconnects applications and LoRa nodes, via gateways that relay messages from and to the wireless network. The network server organizes MAC schedules centrally, while end devices operate in one of three modes: class A (intended for battery-powered devices) is purely producer-driven, best-effort with very limited support for downlink communication; class C is not suitable for the low-power domain; and class B as a tradeoff between both. LoRaWAN networks are subject to collisions [110, 296] and scalability issues [111, 101]. Class B, albeit rarely deployed, is designed to allow periodic downlink communication at low energy, and exhibits reliability issues [263, 382, 316]. It further reveals long downlink latencies. For example, Elbsir *et al.* [102] measured an average waiting time of 44 s at 26 % delivery ratio in a relaxed class B configuration. All those results motivate re-considering the LoRa MAC system design.

### 6.1.2 DSME and LoRa

Motivated by the LoRaWAN deficiencies, we are basing our work on the new DSME-based LoRa MAC design that was introduced by [198]. It has the following key properties that are relevant to this chapter:

In DSME (Figure 6.2), a coordinator emits beacons and initiates a synchronized multi-superframe structure; beacon collisions are inherently resolved for multiple coordinators in reach. Constrained devices (RFD: reduced function device) synchronize to that structure and join the subnet. A superframe is separated into two periods for data transmission: contention-access

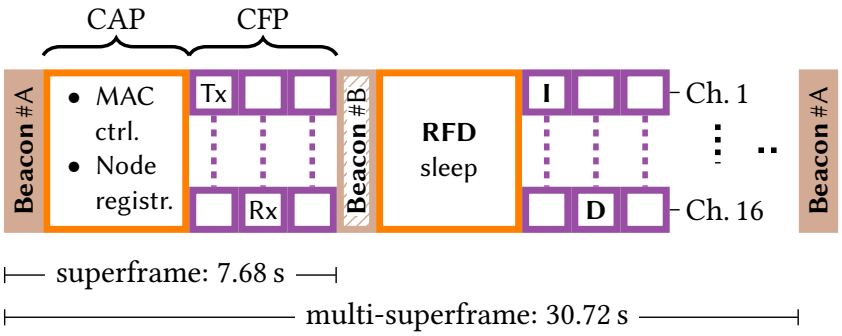



Figure 6.2: Overview of the DSME multi-superframe structure. Perspective of a coordinator. Exemplary schedule for Interest (I) and data (D).

period (CAP), and contention-free period (CFP). This time division facilitates battery powered nodes to enter low-power mode periodically. CFP slots assign unique and frequency-multiplexed transmission resources between nodes to avoid collisions, and provide a deterministic max. latency. Varying slot assignments enable star-, peer-to-peer-, or clustered tree networks. We focus on star topologies.

For the MAC we configure `macSuperframeOrder`: 3, `macMultisuperframeOrder`: 5, and `macBeaconOrder`: 5. This results in a slotframe structure of four superframes per multi-superframe, a beacon interval and multi-superframe duration of 30.72 s (applying the LoRa symbol time of 1.024 ms from Section 6.1.1), and provides 28 time slots · 16 frequency channels = 448 exclusive transmission cells. Other slotframe structures trade off subnet size, throughput, energy, and latency; the latter can increase to over 122 s in certain configurations [11].

## 6.2 Problem Statement

DSME enables an improved LoRa MAC layer design for reliable bidirectional communication, and it can be configured to provide lower latencies compared to LoRaWAN. As such, it is a much better basis for any packet-based higher-layer network stack, including ICN. Still, due to the energy-conservation objectives and the properties of the underlying LoRa PHY layer, even DSME incurs significant delays for interactive communication, based on its multi-superframe structure. These latencies (30 seconds or more) impose significant challenges to any ICN Interest/Data communication, for example, fetching a sensor value from a LoRa sensor, and will require a delay-tolerant communication system.

Superficially, it seems straight-forward to add delay tolerance to ICN, *e.g.,* by simply adding a face implementation for the DTN (Delay Tolerant Networking) bundle protocol [334] or by implementing a delay-aware forwarding strategy on a forwarder. In reality, NDN [413]- and

CCNx [274]-style ICN provides challenges for inter-connecting networks with vastly different RTTs, which is mostly due to the dual functions that Interests provide:

1. Interests and Interest sending rates are central in the transport layer control loop of ICN receiver-driven transport services, *i.e.,* the Interest rate controls the throughput. Interests are used to trigger data transmissions in the first place, and to trigger retransmissions in case no corresponding Data messages have been received within a certain time interval.

2. Pending Interests are temporary state in forwarders that is needed to implement a symmetric forwarding property in ICN, *i.e.,* to record the downstream face that corresponding Data messages should be forwarded on. A secondary function of pending Interest state it to enable *Interest aggregation* – a feature that would prevent multiple Interests for the same Data object to be forwarded on the same path (when there is current pending Interest for that Data object). *Interest aggregation* effectively means *Interest suppression* for all but the first Interest that has been received by a forwarder in a certain epoch – the Interest lifetime in the Pending Interest Table (PIT) of that forwarder.

For achieving a reliable and decently performing communication service, Interest state on forwarders *has to* expire, otherwise Interest retransmission would *always* be suppressed by on-path forwarders that have pending Interest state (and have not received the corresponding Data object yet). There is a time relationship between the Interest lifetime on forwarders and consumer retransmission timers. For good performance, the Interest lifetime needs to be shorter than the retransmission timer.

To cater to delay-prone networks, one could increase both values, maintaining this property. In a heterogeneous network environment (like the Internet), however, it is impossible to decide on "good values". When connecting a high-RTT edge network to a high-speed and low-RTT Internet, both the Interest lifetime and the Interest retransmission timer would need to be adjusted for the end-to-end path RTT. Alternatively, adaptive suppression mechanisms in forwarders (*e.g.,* implemented in NFD [6]) allow for Interest retransmissions in the presence of matching PIT entries. This does, however, still not solve the problem of guessing suitable timeout values for long and vastly different RTT and adopting these timers on every forwarder. Future research and experiments should further investigate different options.

NDN Interests can provide an optional `InterestLifetime` field that allows a consumer to request more suitable Interest lifetime durations (other than the 4 seconds default). We argue that this is not likely to work well in actual deployments:

1. Non-edge, high-speed forwarders are not likely to honor non-standard `InterestLifetime` values for individual Interests to avoid the per-packet performance penalty.

2. In DTN scenarios, RTTs and thus consumer-defined `InterestLifetime` values could be significantly higher than 4 seconds, and a core router may just object to spend memory resources for storing many Interests for a longer time.

3. In DTN scenarios, the RTT may also change unpredictably, depending on caching, opportunistic contacts, new routing state *etc.* so the `InterestLifetime` and the consumer Interest expiration time would have to be adapted constantly, which could introduce brittleness and inefficiency.

It should be noted that ICN in-network congestion control and specific per-forwarder strategies (for example, delay-tolerant forwarding strategies) do not fundamentally resolve these issues because of the interaction with consumers in the non-challenged network and their different understanding of RTTs and retransmission timers. We argue that, instead of guessing suitable `InterestLifetime` values and hoping for all on-path forwarders to honor the corresponding Interest field, it is better to deal with varying and dramatically higher RTTs (*e.g.,* in DTN scenarios) explicitly, with bespoke ICN protocol mechanisms, without interfering with the ICN network layer Interest lifetime.

## 6.3 System Overview

Figure 6.1 illustrates our system model: we want to provide ICN delay-tolerant communication to edge networks, such as a LoRa networks so that hosts on the "regular" ICN Internet can communicate (*e.g.,* request data) with hosts in the challenged LoRa edge network, without requiring Internet hosts and forwarders to apply special `InterestLifetime` parameters and retransmission timers.

Our work is based on three components: (*i*) a mapping of ICN to DSME, (*ii*) gateway node requirements, and (*iii*) delay-tolerant ICN protocol mechanisms for interconnecting challenged networks (including but not limited to ICN/DSME/LoRa networks) to non-challenged networks – aiming for a seamless integration from an application perspective.

### 6.3.1 Mapping of ICN to DSME

DSME provides a contention-access period that is prone to collisions, and a contention-free period (see Section 6.1) requiring a priori slot negotiation. We exclude node association and dynamic slot allocation from this work, as they are orthogonal to the information-centric and delay-tolerant networking aspects. Evolving [198], we simplify the ICN-DSME mapping and use the CAP only for node registration (see below), and the CFP for regular network layer traffic since it guarantees exclusive media access. For the CFP traffic, we implement static scheduling. In bidirectional communication, each Interest slot is followed by a data slot. Consequently, presuming data availability, a request is answered withing the same superframe. For unidirectional data push, a single slot is allocated per node.

### 6.3.2 Gateway Node Requirements

In our system, a LoRa gateway is an application-agnostic, caching ICN forwarder that connects the narrowband LoRa network to the Internet and follows "regular" ICN behavior (*i.e.,* routing) in the upstream direction. Hence, upstream congestion is uncritical since we consider a broadband network as the default deployment. Downstream congestion on the constrained last hop is handled by the buffering gateway. In addition to regular ICN forwarding and caching, the gateway leverages knowledge about expected delays on the LoRa network for adjusting PIT expiry times and `InterestLifetime` accordingly. This PIT state naturally prevents Interest flooding on the wireless medium, as long as it remains active. Caching, as in other ICN scenarios, offloads (re-) transmissions of Interests and Data messages from the wireless link and the constrained nodes. Moreover, the gateway provides these two additional functions:

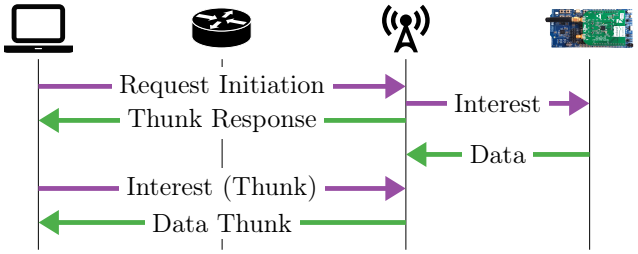
**Node Registration.** LoRa nodes register at the gateway after association, *i.e.,* synchronizing to and joining a network that is advertised by a coordinator. Re-joining a possibly different gateway operates at the order of one (or few) beacon intervals. Nevertheless, it allows for mobile nodes. An overloaded Interest packet by the node indicates its prefix, which establishes a downlink FIB entry on the gateway (see [14]), and the face contains MAC information how to reach that node. Nodes can only serve content under that prefix. On success, the gateway confirms the registration with a data ACK. On a FIB face timeout, *i.e.,* registration expiry, DSME management routines could assist indication (future work).

**Local Unsolicited Data.** The gateway accepts unsolicited ICN Data messages from registered LoRa nodes and acts as a custodian for these nodes. The corresponding content objects are stored in its CS, and the gateway will respond to corresponding Interest messages from the Internet. Caching strategies manage content placement and timeouts for cache eviction. Although gateways are not constrained in memory, least recently used content items are overwritten in case of overflow.

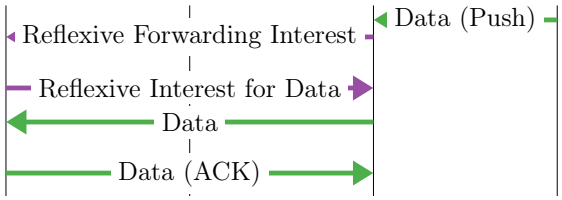
### 6.3.3 Delay-tolerant ICN Protocols

**Delay-tolerant Data Retrieval (Fig. 6.3a).** We want to provide end-to-end ICN communication from an Internet consumer to a LoRa node, *i.e.,* to enable Internet hosts to request arbitrary content objects or to trigger computation in a Remote Method Invocation (RMI) scenario (future work). We leverage the concept of RMI for ICN (RICE [211]) that provides access to static data and dynamic computation results, supporting vastly longer data production/retrieval times. Upon receiving a RICE request initiation Interest, the gateway initiates an Interest message to the LoRa node, as depicted by Figure 6.3a. A so-called "Thunk Response" contains an indication for the waiting time, leveraging link-knowledge about the DSME configuration in the LoRa network.

**Reflexive Push (Fig. 6.3b).** Data generation (*e.g.,* sensor sampling) in the IoT happens sporadically and asynchronously in many cases, which challenges the receiver-driven ("pull")




(a) Delay-tolerant Data Retrieval like RICE.




(b) Reflexive Push like *phoning home*.

Figure 6.3: Delay-tolerant ICN.

ICN-paradigm [63]. The high LoRa latency further motivates a producer-driven data flow in order to avoid periodic polling. This is consistent with [198] who suggest a unidirectional data push for LoRa-ICN. In this scenario, nodes need to register (as described above) before being authorized to push content to the gateway, using the *Local Unsolicited Data* method. This approach assumes a provisioned name as the *phoning home* destination that could be configured when registering the node at the gateway.

We forward these messages to a node on the Internet by leveraging the *phoning home* use case of the *reflexive forwarding* extension to ICN [294]: the gateway sends an Interest to a configured node on the Internet, which triggers a *reflexive Interest* by that node to retrieve the content object (Figure 6.3b).

This approach halves the number of resource-intensive wireless transmissions on the last hop, and doubles the number of available DSME slots per multi-superframe. It should be noted that a next-hop signaling does not introduce new security threats, since a network layer can never prevent a malicious neighbor from transmitting unwanted messages (or jamming) on the local link. The slot-based MAC, however, naturally assists prevention of DDoS, triggered by publishing LoRa nodes. A malicious node can simply be muted by the coordinator (*i.e.,* the gateway), de-allocating its CFP slot.

**Note:** We focus on communication aspects of the protocol mechanisms. Security and corresponding configuration are out of scope for this chapter. Hence, we have slightly simplified the protocol operations in our implementation of these schemes (Section 6.4.2), *e.g.,* we do not use the RICE request parameter retrieval for *Delay-tolerant Data Retrieval*.

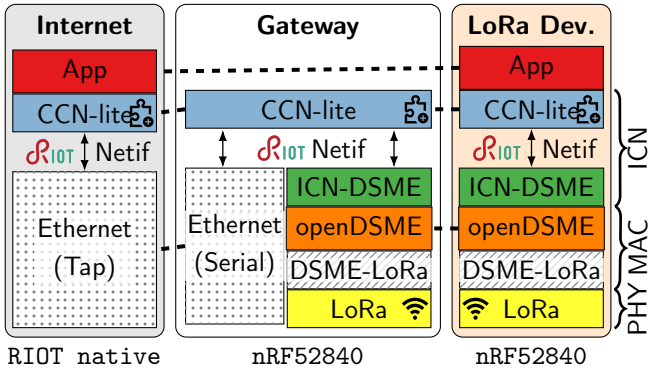



Figure 6.4: LoRa-ICN stacks on different devices with varying resources and network latencies.

## 6.4 Implementation and Deployment

We describe our system implementation in Section 6.4.1 and the protocol implementation in Section 6.4.2.

### 6.4.1 System Setup

We have implemented this system on actual common off-the-shelf LoRa nodes, and we built our LoRa-ICN gateways on the same constrained hardware, to reduce implementation overhead. In a real-world LoRa network (*cf.* LoRaWAN), however, these gateways are not constrained in energy, memory, or processing power and can serve many low-end nodes simultaneously, through radio concentrators. LoRa devices, gateways, and Internet nodes operate the same ICN stack, to overcome incompatibility issues. In the following, we describe the framework (Figure 6.4) that we have created for experimentation.

**RIOT [34].** We base our implementation on RIOT 2022.04. The networking subsystem (namely GNRC) integrates CCN-lite as an ICN stack, which utilizes the generic network interface layer (RIOT Netif in Figure 6.4) to send and receive packets. Currently, wired Ethernet and 802.15.4 CSMA/CA wireless interfaces are available. RIOT supports $>$ 230 IoT boards and a *native* port to execute in a Linux process; it utilizes virtual TUN/TAP interfaces for communication. To build CCN-lite based gateways in RIOT that provide both, a fast wired link and a slow long-range radio, we extend the OS integration layer to utilize multiple network interfaces of varying types, behind an ICN face.

**CCN-lite [372].** Our integration bases on the latest version, checked out by RIOT 2022.04. CCN-lite provides an ICN forwarder implementation and common data structures: PIT, FIB, and CS. A hop-wise retransmission mechanism re-sends a pending Interest after a pre-configured timeout. Note, received Interest retransmissions will be aggregated when hitting an active PIT entry. PIT state expires after a pre-configured `InterestLifetime` value, as usual. We extend CCN-lite by runtime configuration abilities to adjust the PIT- and retransmission timeout,

and the number of retransmissions dynamically. Furthermore, we extend the core forwarder by protocol extensions (🧩) described in Section 6.3 and the mapping to DSME (ICN-DSME in Figure 6.4).

**openDSME [186].** The open access DSME implementation for 802.15.4 radios was ported to RIOT by Alamos *et al.* [12] who also developed an adaptation layer for LoRa (DSME-LoRa in Figure 6.4). Their code is publicly available, albeit not on RIOT upstream. We base our work on their implementation and add interfaces to dynamically control MAC parameters (*i.e.,* ACK request, send period) on a per-packet basis, through the RIOT network interface. The southbound interface utilizes the 802.15.4 radio abstraction API of RIOT.

**LoRa Device.** We deploy the long-range sensor application on common low-power IoT hardware. The Nordic nRF52840 development kit consists of an ARM Cortex-M4 which provides 256 kB RAM, 1 MB flash, and runs at 64 MHz. A SX 1276 LoRa radio shield is attached via pin headers and connects the external radio via SPI. An adjusted transceiver driver implementation exposes the device an 802.15.4 radio, with LoRa specific timing parameters. This facilitates its usage with openDSME. The sensor node is operated as a reduced function device and synchronizes to the DSME multi-superframe, indicated by a coordinator. Afterwards, the node registers its ICN prefix using Interest/Data (see Section 6.3.2).

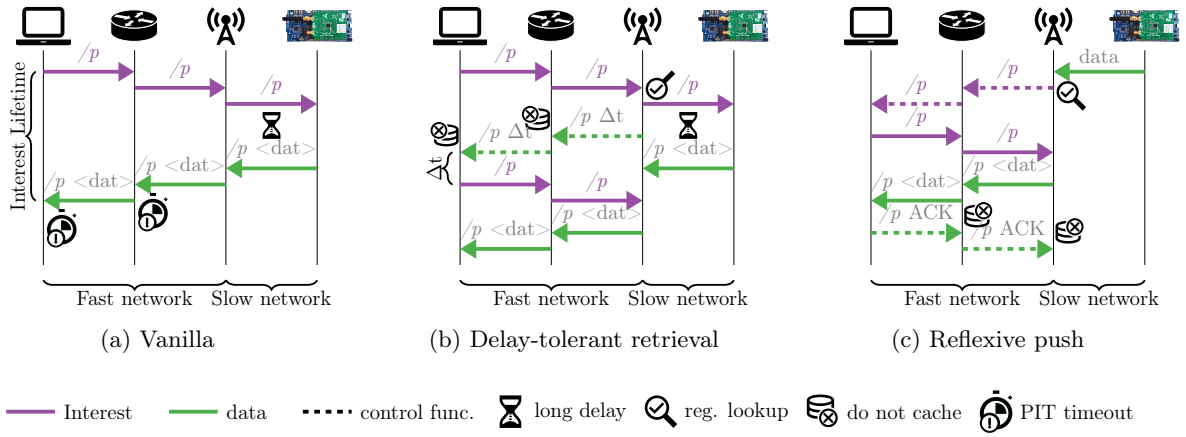
**Gateway.** To reduce implementation overhead, we deploy our gateway on the same hardware as the sensor application. Our gateway acts as a coordinator for LoRa nodes and creates the DSME slotframe structure through the wireless interface. To communicate with a 'fast' infrastructure ICN network in parallel (see forwarder and consumer below), we enable a second network interface; `ethos` is a RIOT specific implementation for Ethernet over serial communication lines. This is required because our experimentation platform lacks Ethernet hardware. Real-world gateways, however, would simply use a gigabit Ethernet link. Our serial device connects to a common Linux based workstation which bridges to a virtual TAP bridge.

**Internet (Forwarder and Consumer).** Nodes on the Internet are emulated by RIOT-*native* instances to utilize the same ICN stack, and connect to the same virtual TAP bridge as our gateway. We deploy two nodes in a line topology, one forwarder and one consumer. Both run in a *Mininet* [268] emulation to enable short link delays of 20 ms and optional link losses on the virtual wire.

### 6.4.2 Protocols for Data Retrieval

We evaluated our system design, comparing its performance to that of regular ICN Interest/Data communication. To that end, we have defined three different data retrieval classes corresponding to Section 6.3.3:

- *Vanilla ICN Request* for regular Interest/Data interactions initiated from a consumer on the Internet;




(a) Vanilla (b) Delay-tolerant retrieval (c) Reflexive push

Figure 6.5: Sequence flows of Interest/Data and ICN extensions between nodes of different time domains. Data flows from LoRa producer to Internet consumer, either initiated by the consumer (6.5a-6.5b) or by the producer (6.5c).

- *Delay-tolerant Data Retrieval* using a simplified RICE exchange initiated from a consumer on the Internet;
- *Reflexive Push* using *reflexive forwarding* and the *phoning home* use case initiated from the producer.

**Vanilla ICN Request.** We assume a regular Interest request from the Internet to the LoRa sensor (Figure 6.5a). The request faces a non-typical long round trip time at the gateway, conflicting with PIT state on forwarders. (*i*) "Regular" forwarders that are not aware of the long delay domain are likely to operate on a fast-network timescale. PIT state that expires before data arrival prevents forwarding on the reverse path. (*ii*) Interest retransmissions are common in ICN, albeit left to transport or application layer implementations. In general, regular ICN-based data retrieval quickly leads to polling and unterminated retransmissions when facing long delays. Two built-in ICN countermeasures are worth discussing: first, `InterestLifetime` dictates the PIT entry expiration time on forwarders. Increasing `InterestLifetime` solves the problem of expired PITs, however, it also requires forwarders to maintain state during the long DSME-LoRa round trip. In addition to occupying PIT memory, this approach affects Interest retransmissions (as a response to timeouts at consumers). Second, common ICN implementations (*e.g.,* RICE, NFD [6]) rely on consumer-based retransmissions (contrasting in-network retransmissions). This *requires* PITs to expire fast, otherwise, a retransmission will be suppressed.

**Delay-tolerant Data Retrieval.** We have implemented the interaction from Section 6.3.3 by adding server logic to the link-aware gateway that is triggered by the reception of corresponding Interest messages from consumers in the non-challenged Internet (Figure 6.5b). The gateway performs three major actions after an incoming Interest: (*i*) It first checks for a registered LoRa

node that falls under the requested prefix, in its FIB. (*ii*) On a missing FIB entry, it immediately returns a data NACK. (*iii*) On success, it forwards the Interest as per regular forwarding using the FIB face towards the LoRa node. On forwarding, the gateway replies with a distinct data NACK (we call it WAIT) which contains an estimated data arrival time. A gateway can provide accurate estimates in the future, using its knowledge of the DSME configuration upfront, the internal scheduler state, as well as the current traffic load (queue length). This data packet satisfies the initial Interest, corresponding in-network state, and terminates potentially inappropriate ICN-based retransmissions. The estimated data arrival time enables the consumer application to set an appropriate retry timer, without the need for specific producer knowledge and varying long delays introduced by DSME-LoRa. NACK/WAIT data packets in (*ii*) and (*iii*) must not be cached, though, to prevent serving a subsequent request of the same name from the CS. Finally, after a repeated Interest request, the data item is likely served from the gateway.

**Reflexive Push.** Our protocol flow (Figure 6.5c) implements the second interaction from Section 6.3.3. It suggests two nested Interest/Data exchanges. After successful content placement on the gateway, using *Local Unsolicited Data*, this one indicates data by sending an Interest packet that contains the data name, to the consumer. An additional packet indicator triggers the establishment of a temporary downlink FIB entry on forwarders for that specific name, which points to the incoming face. The consumer can return a *reflexive Interest*, requesting the announced data; it follows the previously established FIB path. Data is served from the gateway cache as usual, satisfying PIT state on the reverse path, and additionally removes the temporary FIB entries. An optional final data ACK terminates the initial Interest request.

## 6.5 Evaluation

We describe experiment configurations in Section 6.5.1, measurement results for protocol performance in Section 6.5.2, results from our analysis of communication overhead in Section 6.5.3, and system overhead of the protocol stacks in Section 6.5.4.

### 6.5.1 Experimental Setup

We conducted five experiments (comparing our two schemes described in Section 6.3 with three *Vanilla ICN* variants):

**Vanilla (1)** Baseline scenario with unchanged ICN and common parameter settings.

**Vanilla (2)** Delay-aware consumer with extended `InterestLifetime` and retransmission interval.

**Vanilla (3)** Like **(2)**, additionally forwarders observe the long `InterestLifetime` and set their PIT timer accordingly.

Table 6.1: Scenario and parameter overview including four measured nodes. (Abbreviations: INR=In-network retransmission, CR= Consumer retransmission, ✘= not applicable).

| Scenario | | Cons. | | Fwd. | | Gw. | | Node | |
|---|---|---|---|---|---|---|---|---|---|
| | | ⏱ | ↻ | ⏱ | ↻ | ⏱ | ↻ | ⏱ | ↻ |
| **Vanilla (1)** | INR | 4 | 3:1 | 4 | 3:1 | 60 | 0:0 | 60 | ✘ |
| | CR | 4 | 3:1 | 4 | ✘ | 60 | 0:0 | 60 | ✘ |
| **Vanilla (2)** | INR | 60 | 3:15 | 4 | 3:1 | 60 | 0:0 | 60 | ✘ |
| | CR | 60 | 3:15 | 4 | ✘ | 60 | 0:0 | 60 | ✘ |
| **Vanilla (3)** | INR | 60 | 3:15 | 60 | 3:1 | 60 | 0:0 | 60 | ✘ |
| | CR | 60 | 3:15 | 60 | ✘ | 60 | 0:0 | 60 | ✘ |
| **Delay-tolerant retrieval** | INR[1] | 4 | 3:1 | 4 | 3:1 | 60 | 0:0 | 60 | ✘ |
| **Reflexive-push** | INR | 4 | 3:1 | 4 | 3:1 | 4 | 3:1 | ✘ | 0:0 |

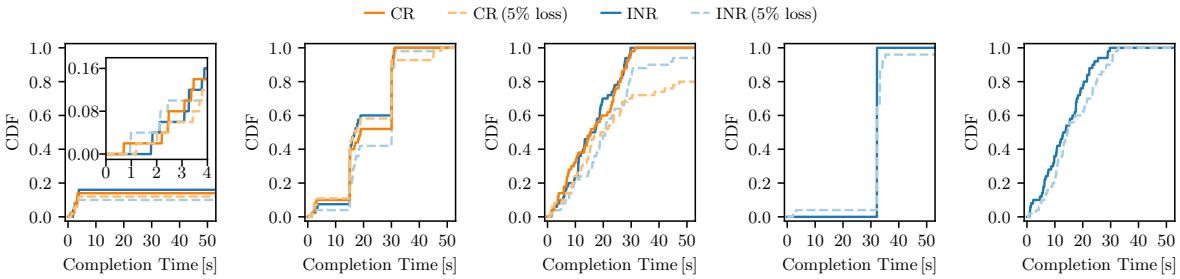
⏱ PIT timeout [s] ↻ Retransmission attempts and timeout [#:s]

[1] Additional retry based on WAIT instruction on first request.

**Delay-tolerant retrieval** Gateway acts as a special proxy for long-delay producers and returns a distinct re-try instruction on first request.

**Reflexive push** Producer initiates a transaction by pushing data to gateway CS which triggers a reflexive Interest/Data interaction for retrieving content.

In our experiments, we use unique content names, prefixed with a LoRa node ID and incremental local object counters. Data contains either a random integer value or an ACK, NACK, or WAIT instruction with a time hint. This fixed size scheme leads to a frame size of 31 Bytes for Interest and 36 Bytes for data, which leaves headroom to the maximum frame size of 127 Bytes. Longer packets, however, could be compressed [148] and fragmented [231] in the future. Every content item is requested/indicated once, with an average interval of one minute (60±10 s uniformly distributed). For a fair comparison between consumer- and node initiated traffic, we produce sensor data on the LoRa node after an incoming Interest. Data returns during the subsequent CFP slot within the same superframe (compare Section 6.3.1). Our measurements include: (*i*) completion time, *i.e.,* the delay between issuing a transaction and data arrival at the consumer; (*ii*) resilience, *i.e.,* the rate of successful transactions; (*iii*) protocol overhead, *i.e.,* the number of transmitted packets per content item. Thereby, we deploy an idealized scenario with 0% – and the case for 5% link loss on the Internet emulation.




(a) Vanilla (1) (b) Vanilla (2) (c) Vanilla (3) (d) Delay-tolerant retrieval (e) Reflexive push

Figure 6.6: Time to content arrival with long producer delays. Vanilla ICN in varying configurations and our extensions employ in-network retransmissions (INR) or consumer retransmissions (CR), and we vary the link loss.

Table 6.1 summarizes our parameter settings. All but the last scenario require the gateway and node to lift the PIT expiration time to the long delay domain. We conservatively chose 60 s which reflects ≈ two times the multi-superframe duration of the MAC (compare Section 6.1.2). Retransmits on the LoRa hop are disabled since we utilize exclusive CFP resources.

For Vanilla ICN, we distinguish the case with in-network retransmissions (INR) and consumer-based retransmissions (CR), with different PIT timeout behavior. Our **Vanilla (1)** configuration assumes that Internet nods are unaware of the long delay domain. Hence, we set a PIT expiration time of 4 s according to default settings of the common NFD implementation [6] and enable three network layer retransmits, each after 1 s, which reflects the initial round-trip estimation of TCP [301]. In **Vanilla (2)**, a consumer is aware of long producer delays, hence, we set `InterestLifetime` and PIT expiration time to 60 s as well, and adjust the retransmission interval to 15 s. Forwarders do not adopt the long timeout value. In **Vanilla (3)** the forwarder adopts the `InterestLifetime` value of the incoming packet and sets its PIT expiration time accordingly, *i.e.,* to 60 s. This does not change its retransmission behavior in the INR case, however. We present two alternative solutions: **Delay-tolerant retrieval** gets along with 'short' Vanilla (1) parameters and utilizes INR. **Reflexive push** inverts the original ICN semantic and consists of two nested Interest/Data flows that utilize 'short' time parameters analogously.

### 6.5.2 Completion Time and Resilience

Figure 6.6 presents the cumulative distributions of completion times of successful transactions. These values are mainly affected by the multi-superframe duration of the MAC (30.72 s) which dictates the maximum latency of a unidirectional long-range transmission between. Data losses result in infinite completion times, hence, the end value of each graph inherently reflects its success ratio.

**Vanilla (1) (Fig 6.6a).** 10–16% of requests are successful and finish in less than the PIT

timeout of 4 s. This is the case for Interest that happen to arrive at the gateway short before a DSME transmission slot occurs. Link losses further drop the success rate by 2–6%, but different retransmission pattern do not provide a significant effect.

**Vanilla (2) (Fig. 6.6b).** Completed transmissions in <4 s resemble properties of the Vanilla (1) case. Steps at 15 s indicate the poll interval of the consumer, which recovers losses from long DSME-LoRa delays. This requires, however, that forwarder PIT state expires fast (here 4 s) to prevent *Interest aggregation*. Losses delay the completion and are compensated faster with INR overall (≈ 32 s), though, CR recovers 20 % more requests on the first retry. Conversely, 10 % of the requests require a third retry with CR, to complete successfully (≈ 45 s). A comparison of CR with and without loss reveals a diverse picture. Here, the lossless case surprisingly satisfies fewer requests after the first retry, which is an effect of randomized experimental requests.

**Vanilla (3) (Fig. 6.6c).** Cases without link loss require 32 s at max. (multi-superframe duration) to retrieve all content, which directly reflects the delay distribution of the DSME-LoRa MAC. The long PIT state on both consumer and forwarder allow data forwarding whenever it is ready, reflecting the case for soft-state subscription by a long-lived Interest [67]. Link losses, however, demonstrate the drawback of this approach. CR prevent effective loss recovery while PIT sate is active on the forwarder, and drop the delivery rate to 80 %. INR recover most losses and result in 94 % delivery. This approach only performs well under the assumption that (*i*) every forwarder adopts the long PIT timeout, and (*ii*) content can be retrieved within that time.

**Delay-tolerant retrieval (Fig. 6.6d).** Requests finish in almost exactly 32 s in the lossless case, which is the returned WAIT time of the gateway after the first request of a content item. This static worst case value could be reduced with a latency estimator model on the gateway, allowing for targeted completion times. The gateway retrieves content from the node during WAIT, and caches it. A subsequent request of that item is answered from the gateway CS. Additional link losses are mostly recovered by INR and perform similarly to the lossless case, however, two effects are noteworthy: (*i*) ≈ 5 % of the requests finish below 3 s. A loss of the first data packet, which contains a WAIT instruction, triggers an INR which is already satisfied by the gateway. (*ii*) ≈ 8 % of the requests are not satisfied. Our implementation uses a short circular list of future requests to re-issue, which avoids (larger) PIT state over long time. In the loss case, entries stayed longer in the list and got overwritten occasionally, leading to unrequested data. In practice, the list should be provided with timeout values and dimensioned according to traffic load.

**Reflexive push (Fig. 6.6e).** Transactions finish in max. 32 s (multi-superframe duration) with 100 % success. Completion times reflect the delay distribution of DSME-LoRa, similarly to the Vanilla (3) scenarios. Herein, the additional round trip of a nested double Interest/Data flow has a negligible overhead when directed towards the fast network. Thereby, losses are smoothly recovered by INR, at minimal time overhead. Contrasting Vanilla (3), this approach

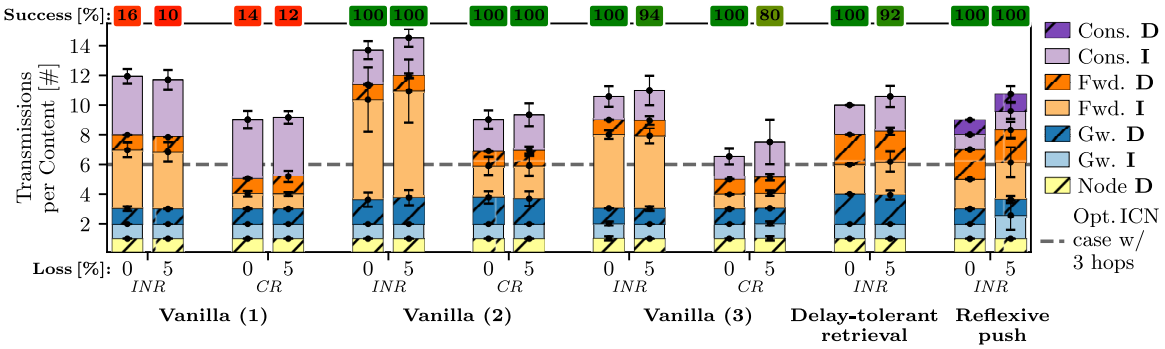



Figure 6.7: Transmissions per content item (protocol overhead) separated into consumer (Cons.), forwarder (Fwd.), gateway (Gw.), and LoRa node. Vanilla ICN in varying configurations and our extensions employ in-network retransmissions (INR) or consumer retransmissions (CR), and we vary the link loss. Lossless ICN transmission via 3 hops is indicated by the dashed line.

works with arbitrary (producer) delays and forgoes the need to adopt long PIT timeout values on Internet nodes.

**Findings.** Expired PIT state on the reverse path is the prevalent obstacle with vanilla ICN and prevents round tips >4 s, which renders the baseline scenario unusable in this domain. Application-aware consumers overcome long delays, however, the performance heavily depends on the (arbitrary) choice of a poll interval and is susceptible to *varying* delays. Increasing the `InterestLifetime` on the complete forwarding path, instead, is challenging. (*i*) We cannot expect real forwarders to blindly adopt arbitrary PIT timers. (*ii*) Without in-network retransmission in place, long-lived PIT state harms reliability. The Delay-tolerant retrieval case overcomes requirements of long PIT state and blind polling. It thereby relieves Internet nodes and applications from knowledge of the (variable) long time domain. Consumer implementations become more complex, therefore. A reversed transaction flow with Reflexive push facilitates efficient, reliable, and 'timely' transactions.

### 6.5.3 Communication Overhead

Figure 6.7 quantifies the protocol overhead for every node and scenario (cf. Section 6.5.1) and shows the number of transmitted Interest and Data packets per requested content item as well as the success rate, replicated from Figure 6.6. In a three hop network, an optimal ICN request-response requires six packets, as indicated by the dashed line. Recall that all scenarios but *Reflexive push* lift the PIT timeout on the gateway and disable network layer retransmissions on the LoRa link, to preserve sparse resources. Consequently, gateways only transmit one Interest towards nodes that respond with one data packet per request.

**Vanilla (1).** These scenarios reveal notable overheads by futile retransmission, regardless

of link loss. Up to two times as many packets are transmitted, compared to the ideal case, with little overall success. With INR, both forwarder and consumer transmit at maximum (4 Interests/content), while CR keeps forwarder overhead low (1 Interest/content). Interests are aggregated as long PIT state persists. Standard retransmit intervals cannot cope with long delays.

**Vanilla (2).** INR reveal the highest overhead among all scenarios (15 transmissions), sending requests at two timescales. Every consumer Interest is forwarded *and* retransmitted by the forwarder, regardless of the long delay of the producer. In contrast, CR overhead ($\approx 9$ transmissions) is on par with Vanilla (1) CR but satisfies all requests, without blind forwarding. Hence, PIT timeouts $<$ consumer poll intervals that operate at the prevalent delay domain are a viable option for the conventional ICN paradigm. Short-lived PIT state cannot prevent duplicate data transmission by the gateway, though, when Data faces expired PIT state on a forwarder.

**Vanilla (3).** INR recovers link losses, while a 'sufficiently' long PIT expiry time prevents consumer-based retransmissions. The CR case (without loss) thus operates with little overhead ($\approx 7.5$ transmissions) but is not vital due to high sensitivity to link loss.

**Delay-tolerant retrieval.** Our approach generally increases the required transmissions per content, introducing a second round trip between gateway and consumer. Hence, it performs optimal in the lossless case by transmitting 10 packets: 2xInterest/Data on fast nodes, and 1xInterest/Data on the LoRa link. INR marginally increase the overhead. The total overhead compares to Vanilla (2) with CR, however, it surpasses blind polling.

**Reflexive push.** Our second approach inverts the flow direction and introduces a second round trip between gateway and consumer as well. In contrast to Delay-tolerant retrieval, however, only a single LoRa transmission is required to place producer content in the gateway cache. This producer oriented optimization results in an optimal number of 9 transmissions per content, reflected by the lossless case. INR increases up to 10.5 transmissions (avg) on loss. Data ACKs by the consumer are optional and terminate the initial Interest of the gateway. Omitting these packets is principally possible to reduce transmissions, however, this conflicts with INR.

**Findings.** Delay-tolerant retrieval and Reflexive push are robust, operationally efficient, and can tolerate varying delays of the DSME-LoRa MAC. In contrast, vanilla ICN requests suffer from long and unpredictable delays. Naive consumer polling is an inefficient but viable ICN-idiomatic alternative, provided that Interests expire on the forwarding path and polling intervals are set in agreement with practical delays.

Table 6.2: Energy consumption per multi-superframe and lifetime for the protocols under consideration.

| Protocol | Energy [mJ] | Lifetime [d] |
|---|---|---|
| **Vanilla ICN request** | | |
| w/o MAC | 1247.46 | 10 |
| w/ MAC | 51.42 | 230 |
| **Delay-tolerant data retrieval** | 51.42 | 230 |
| **Reflexive push** | 30.83 | 384 |

### 6.5.4 System Overhead

We evaluate the resource overhead of our protocol stack and focus on the battery driven LoRa device, since gateways and Internet nodes are not resource-constrained and remain unchallenged by common LoRa traffic.

**Energy Consumption.** We present the energy consumption per multi-superframe in Table 6.2, as well as the corresponding nodal lifetimes when operated from an off-the-shelf AA alkaline battery (2800 mAh). Our results are based on extensive measurements performed in [11], which quantify the energy consumption for passive and active periods of the DSME-LoRa superframe structure. Radio operations dominate consumption, *i.e.,* wireless transmission and (idle) reception. To confirm this observation, we also measure the active CPU time throughout our experiments, which is as low as $\approx 0.25\,\%$ for all protocols on the constrained node, and around $0.3\,\%$ on the gateway. The latter increases with growing network sizes.

Vanilla ICN request values include the alternative operation without a MAC (ignoring wireless interference), which strongly motivates the choice of a duty cycling MAC from the energy perspective. Without duty cycling, the lifetime is limited to 10 days. Enabling the MAC reduces the energy consumption by two orders of magnitude, which leads to a lifespan of 230 days in the vanilla ICN request and delay-tolerant data retrieval case, assuming that the gateway shields LoRa devices effectively from retransmits. Reflexive push almost halves the energy consumption due to unidirectional transmission, which further increases the lifetime to more than a year.

**Memory Requirements.** Our network stack is runtime configurable to operate the three protocols for data retrieval (Section 6.4.2). Hence, the firmware image is the same for all configurations and requires 143 kB in ROM (text + data segment) and in 19 kB RAM (bss + data segment), almost half of which is occupied by openDSME. The remaining RAM (256 kB on nRF52840) is reserved for dynamic runtime memory allocation (heap). Both openDSME and CCN-lite utilize `malloc`, and we track the combined heap statistics which ranges between 6–8 kB in all experiment runs. Thus, our LoRa-ICN stack can even be deployed on much smaller IoT hardware.

## 6.6 Related Work

**Advancing LoRa(WAN).** To overcome limitations of the centralized LoRaWAN architecture, multi-hop extensions for LoRa [367, 130, 48, 80] have been proposed. These are orthogonal to our work since we focus on single-hop topologies.

Contention-based [233, 290] and scheduled MAC layers [406, 418, 168, 184] for LoRa indicate performance improvements compared to LoRaWAN. Alamos *et al.* [12, 11] re-utilize IEEE 802.15.4e DSME (Deterministic and Synchronous Multi-Channel Extension) [174] to coordinate LoRa radios, with few modifications to the radio configuration. Fixed time-slotted DSME paired with low data rates increases latencies even further, though. In this work, we enable LoRa to run a robust DSME-based MAC layer with latencies that we are able to handle.

RFC 9011 [124] specifies Static Context Header Compression and Fragmentation (SCHC) for IPv6 over LoRaWAN. We agree that compression and fragmentation are crucial, but do not address the latency issues for transport protocols. Also, SCHC does not fix the underlying MAC, which is prone to collisions and depends on network server scheduling.

**ICN and the IoT.** The IoT benefits from ICN [33, 307, 258, 339, 338, 142, 26, 410]. An important observation in prior work is that IoT scenarios require the adaptation of the MAC layer to prevent unnecessary broadcast and preserve energy resources [193]. Current analyses either base on 802.15.4 CSMA/CA [142], requiring receivers to be always on, or 802.15.4e TSCH [163], allowing for intermittent device sleep.

NDN over LoRa was introduced in [204, 243] which required permanent powering of the nodes, depleting the battery. Unfortunately, latency analyses have not been considered. Recent work [243] shows the need for a MAC protocol due to high collisions even when deploying only few LoRa nodes.

A system design for ICN over DSME-LoRa is proposed in [198]. Based on simulations, the authors find latencies at the order of tens or hundreds of seconds. In this chapter, we close the gap and present a solution to handle these high delays and thus enable common, inter-network IoT deployments.

**Delay-tolerant ICN.** Another ICN application domain that is challenged by long delays are satellite networks. Siris *et al.* [348] find that hop-wise transfer and caching help to increase performance in such networks. They consider an Interest as a long-lived subscription. In contrast, Kumari *et al.* [217] argue that NDN is not viable in satellite scenarios, due to inefficient polling. This is in line with our experimental results. To reduce long delays and needless retransmissions during satellite handovers, the adjustment of the forwarding path is proposed [238]. This solution requires a signal after connecting to a new satellite.

Carofiglio *et al.* [66] exploit link signaling to indicate some kind of loss to trigger a PIT lookup and eventual retransmits, reducing RTTs and redundant retransmits. LoRa lacks such signaling capabilities. We incorporate link awareness in our proposed DSME-LoRa gateway.

Kuai *et al.* [213] propose delay-tolerant NDN forwarding for vehicular networks. Fundamen-

tally, neighbored nodes overhear surrounding traffic and adjust their retransmission procedure based on directional network density. In simulations, the authors assume a relatively high PIT timeout of 50 s. To prevent large PIT tables due to unnecessary long-lasting entries, NACK data packets can include instructions when to retransmit an Interest [270, 76]. Similarly to delay-tolerant networking with NDN [235], the IoT requires a mechanism apart from pure request-response.

**Producer-initiated ICN.** Burke *et al.* [63] propose push-based sensor data dissemination, accepting names within a distinct namespace on the consumer. Gündogan *et al.* [146] evaluate name indication that triggers a conventional Interest request on the consumer. Król *et al.* [211] introduce a nested 4-way handshake to enable RMI use cases based on ICN principles, and analyze drawbacks from long latencies. This approach is in line with *reflexive forwarding* [294]. We exploit both push and indication concepts in our evaluation.

## 6.7 Conclusions

Interconnecting networks with vastly different RTTs is challenging for any non-trivial communication system, including ICN. ICN, unlike other frameworks, however, has the unique potential to enable robust communication to nodes in challenged edge networks without requiring application layer relays. In conjunction with an OS-level implementation of ICN (and extensions), DSME, and LoRa, our two protocol mechanisms for Internet consumer-initiated and LoRa producer-initiated communication exhibit high reliability and targeted completion time (compared to Vanilla ICN) when applied to the delay-prone regime. Despite an additional round trip, our evaluations show low overhead of these approaches, by overcoming redundant polling. We leveraged recently proposed gateway behavior (like RICE) and ICN protocol extensions (*reflexive forwarding*), the latter of which serves many other use cases beyond *phoning home* and could be considered a useful standard ICN feature.

# Part II

# System-level Security on Constrained Embedded Devices

# Chapter 7

# Motivation and Problem Statement

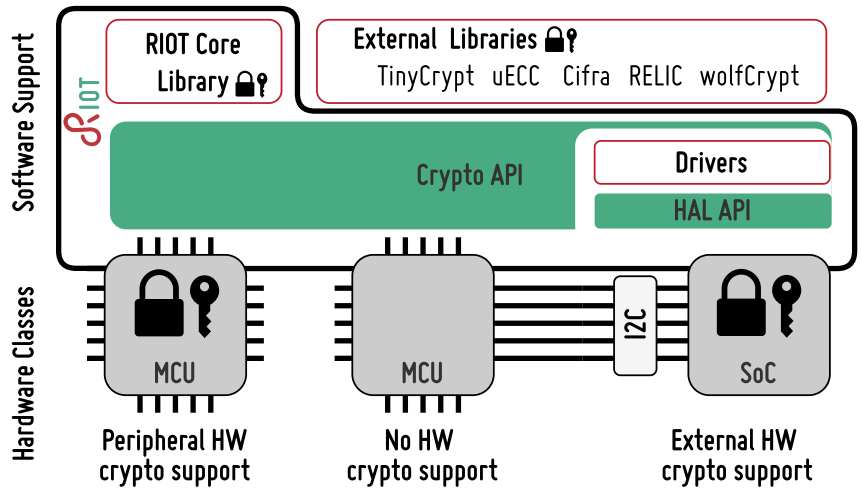



Figure 7.1: The software support layer of RIOT integrating crypto-peripherals, external crypto-devices, and crypto-libraries using a common crypto API.

## 7.1 Heterogeneous Crypto-hardware and Software in the IoT

**Cryptographic Backend Categorization.** Figure 7.1 presents three common options to enable crypto-operations in the constrained IoT, and our system integration that we utilize for aligned performance comparisons in Chapter 8. We categorize cryptographic backends into three groups. (*i*) Cryptographic software libraries (*e.g.,* RIOT Core, TinyCrypt, uECC) that try to cope with embedded constraints. These libraries commonly lack crypto-hardware support for portability reasons. (*ii*) Microcontrollers that include a peripheral for cryptographic acceleration, which enhances the crypto-performance and reduces energy requirements. Manufacturer SDKs commonly support their own chipsets but reduce flexibility towards a vendor lock-in. (*iii*) External crypto-devices (secure elements) that connect to the microcontroller using a com-

munication bus. A separate chip acts as an isolated crypto-processor and key storage which requires additional energy, but allows for sleep cycles of the main processor during operation.

**Operating System Support.** IoT nodes of class 0 are programmed with bare metal firmware due to extremely limited hardware resources. Class 1 nodes more and more utilize operating systems [98] instead of bare metal firmware, to keep applications portable while gaining a thorough hardware support. A key motivation of this work is to make heterogeneous hardware components uniformly accessible for both security protocols and applications, which enables usable security in the low-end IoT. An operating system should provide transparent access to the available hardware without sacrificing performance nor functionality, by contributing driver code below the hardware abstraction layer, or software libraries as a fallback. This poses challenges on the system integration, since crypto-hardware features are largely heterogeneous and range from extended instruction sets that can complement software implementations, to complete hardware-implementations of popular algorithms such as AES, ECDSA, *etc.* Heterogeneous hardware features demand for a feature model in order to compile targeted firmware images with optimal hardware support and domain compliant memory footprint. Varying levels of crypto-hardware assistance require a layered software design at different levels, which needs to be reflected by the feature model.

**Performance Comparison.** Cryptographic primitives should be optimized in the IoT and utilize the constrained hardware most efficiently—including possible crypto-extensions. Chapter 8 introduces our crypto-subsystem in the IoT operating system RIOT, which integrates different types of crypto-backends. A comparative measurement study of five software libraries and four hardware platforms quantifies the performance of basic symmetric and asymmetric cryptography on commodity platforms with and without cryptographic hardware. We further evaluate the impact of different device driver design options.

## 7.2 Revisiting Randomness Generation on Embedded Devices

In the literature, a "random sequence" is often referred to by the definition of *D. H. Lehmer*:

> *"A random sequence is a vague notion [...] in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests, traditional with statisticians."*
> —— Lehmer [229]

**Application of Random Numbers.** Random numbers are required by various IoT applications and components of an operating system, in the context of security as well as in basic system operations. Figure 7.2 presents an overview of the typical use cases of random numbers in an IoT system.

System services, *e.g.,* MAC protocols, apply random delays to avoid interference. IoT applications such as environmental monitoring, automated machinery, machine-to-machine communication, and incident reporting utilize random input. The need for random numbers increases

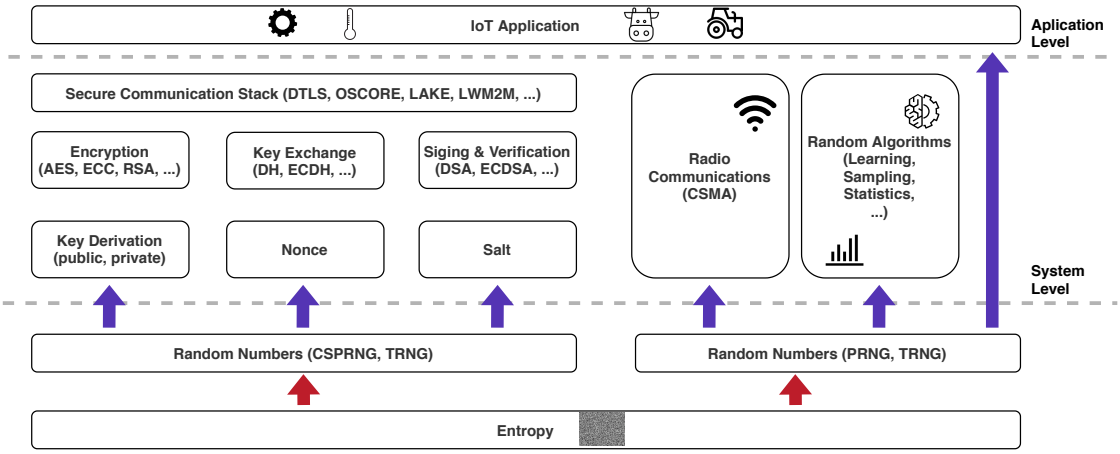



Figure 7.2: The role of random number generation in IoT applications.

even further with the advent of Artificial Intelligence (AI) in the IoT, which shows promise to improve existing use cases as well as to create completely new application scenarios on small and cheap devices. Machine learning at the Internet edge, for example, may be used to pre-process data, to improve physical measurements, and reduce network load. This involves randomized algorithms and large-scale random sampling running on constrained IoT devices.

**Random Numbers for Security.** Random numbers are required in almost all security primitives to generate or perpetuate secrets such as encryption keys or cipher streams. Security protocols and securing communication layers pose strong requirements on the secrecy of random numbers for guaranteeing confidentiality, integrity, and privacy. The left part of Figure 7.2 displays the composition of security components, which are assembled in a crypto-stack. Security protocols (*e.g.,* IPsec, DTLS, OSCORE) rely on keys and nonces to encrypt, sign, or validate signed network packets. Naturally, they must resist prediction and the random input must consequently be unpredictable as well. Such demands require an entropy source in the system. Having a robust security system in place will also enable completely new security applications benefiting from IoT devices, *e.g.,* to implement off-grid blockchain transactions or further variants of distributed ledger technologies.

**True vs Pseudo-random Number Generation.** Ideally, true random number generators (TRNGs) create values with maximum entropy. True randomness is hard to enforce, and gathering of entropy consumes system resources that are sparse on IoT devices. True random sequences are generated from random physical processes such as thermal noise, manufacturing inaccuracies, or crystal drift. Current personal computers extend such sources to sound or video input, disk drives, user keystrokes, and more as proposed by the IETF [1]. These interfaces do not necessarily exist on embedded IoT devices. There may be complementary sources of random input such as antenna noise or sensor measurements. Still, collecting random values is difficult and resource-intensive because underlying processes are slow or do not output continuously. Fur-

thermore, operating additional hardware components increases the energy consumption which challenges battery driven nodes.

Pseudo-random number generators (PRNGs) are deterministic algorithms. Given a random seed as input, PRNGs output sequences that look random to any analysis method that runs on the sequence without knowledge of the input value. If properly seeded, these generators expand a comparably short seed value into a long sequence of (pseudo-)random numbers [91] while discontinuing to depend on (physical) random processes. Using a PRNG also reduces the attack surface for adversaries with physical device access who try to tamper with the environmental conditions. Cryptographically secure PRNGs (CSPRNGs) provide unpredictable random numbers suitable for security purposes, given that their seed values are kept secret. CSPRNGs are based on a proof that defines hard to solve challenges which an attacker is unable to solve and to break. In order to be fully unpredictable, though, the CSPRNG requires initial seed values with maximized entropy. Guessing these values must be infeasible for an attacker, which means that it terminates in time faster than $2^{128}$ steps (recommended security strength [39] at the time of writing).

**Requirements for Low-end Devices.** Producing random sequences on low-end IoT nodes faces particular challenges. As a frequently invoked system service, a random function should be frugal in resources. Neither should it delay algorithms, procedures, or protocols, nor should its state overhead diminish the scarcely available main memory, nor should its energy demands affect the system. Many random algorithms deployed in regular computer systems or clouds are too demanding and violate these constraints. Hence, it is important to identify random number generators that comply with the IoT constraints, which is a major objective of Chapter 9.

Requirements on the secrecy of random numbers on constrained devices are similarly strict as with server machines that are easily available to adversaries. Still, cryptographic operations involved in CSPRNGs introduce a notable overhead in processing time and energy consumption on constrained IoT devices that often run on small batteries. Exploiting this, an adversary who manages to trigger random operations on the IoT device may even run energy depletion attacks. Hence, it is of particular importance to choose algorithms and implementations of CSPRNGs that minimize resource loads while standing up to the standards of unpredictable secrets.

**Evaluation of Random Number Generation.** Chapter 9 provides systematic testing and performance evaluations of hardware and software random number generators on IoT platforms that are operated with the RIOT operating system. First, we analyze off-the-shelf random number generating hardware using common statistical test suites that detect functional weaknesses, and multiple of performance benchmarks to compare the resource overhead on constrained devices. Second, we perform analogous tests and evaluations for pseudo-random number generating software. Comparing hardware and software properties and their resource consumption leads to a conclusive discussion on how to combine and jointly deploy the different hardware and software components from the perspective of an IoT operating system.

## 7.3 Hardware-intrinsic Sources of Entropy and Uniqueness

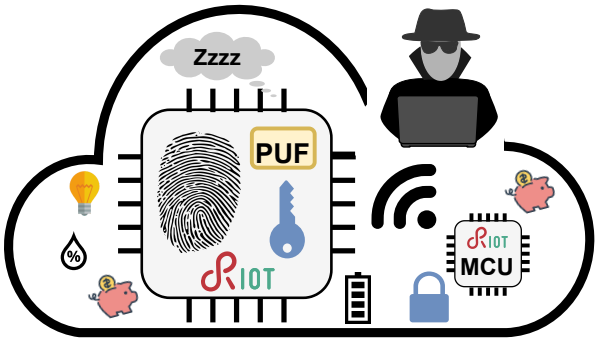



Figure 7.3: PUF security services provided by an operating system enable lightweight crypto-operations on low-cost hardware in the IoT.

High entropy seeds for secure random number generation (*cf.* Section 7.2) and secure hardware identities form the minimal set of primitives that bootstrap the cryptographic subsystem needed for protecting basic services of networked nodes (see Figure 7.3). These numbers must remain secret to prevent information leakage of past and future transactions, and require resistance against readout or tampering. Physical unclonable functions are a promising solution to provide (*i*) random variations on one device, and (*ii*) unpredictable secrets between devices that become reproducible by excluding the variations from (*i*). They utilize intrinsic hardware variations instead of adding hardware security modules (*e.g.,* secure elements) which would increase device cost.

**SRAM as a PUF.** A prevalent type of PUF input is SRAM. Manufacturing processes introduce random variations in the silicon of transistors that construct a memory cell. After powering up the SRAM, it provides a digital fingerprint based on the patterns of uninitialized memory, which facilitates seed and key generation thereof. SRAM is available on almost every IoT platform and can be exploited without additional hardware. This makes the technology particularly attractive for low-cost devices. Secret values are generated only during system startup and consumed quickly after, to lower the risk of a compromise. Consequently, SRAM secrets remain absent during regular node operations, and vulnerable data is absent while a node sleeps, the prevalent state of a battery-driven embedded device.

**Bias and Aging.** There have been long-standing concerns that the physical layout of SRAM as well as hardware aging [248] may introduce systematic biases [251, 216]. Quantifying these statistical effects requires a comparative analysis between large quantities of devices [397], which we contribute in Chapter 10. Our large-scale evaluation shows a localized bias for certain bits in the SRAM response. An attacker who tries to predict this pattern by using a large number of measurements from similar devices could reach an advantage in guessing the bit values at

certain positions. We quantify the remaining entropy of secrets derived by these biased pattern and identify secret generation schemes that are able to mitigate this weakness.

We further analyse hardware aging by extensive measurements on nodes that naturally aged in the real-world environment of an open access testbed. We find address specific wear-out effects that link to past experiment executions. These findings shall motivate testbed operators as well as software developers to invoke anti-aging strategies to their firmware.

**OS-level Evaluation.** In Chapter 10, we design and evaluate PUFs for the multi-purpose operating system RIOT. To the best of our knowledge, a consistent PUF integration into a commodity IoT operating system is yet missing, even though IoT deployments increasingly rely on some (open source) operating system. We perform the first SRAM evaluation with a statistically significant sample size of more than 700 devices in the FIT IoT-LAB testbed. In a baseline evaluation of our OS-level solution, we quantify the uniqueness of SRAM PUF generation. Our second analysis concentrates on the quality of random seed and key generation, as well as its performance overhead. A subsequent security analysis reveals that the SRAM PUF is secure under moderate attacker assumptions.

# Chapter 8

# Analysis and Integration of Cryptographic Backends


## Abstract

In this chapter, we contribute a comprehensive resource analysis for widely used cryptographic primitives across different off-the-shelf IoT platforms, and quantify the performance impact of crypto-hardware. This work builds on the newly designed crypto-subsystem of the IoT operating system RIOT, which provides seamless crypto support across software and hardware components. Our evaluations show that (i) hardware-based crypto outperforms software by considerably over 100 %, which is crucial for nodal lifetime. Despite, the memory consumption typically increases moderately. (ii) Hardware diversity, driver design, and software implementations heavily impact resource efficiency. (iii) External crypto-chips operate slowly on symmetric crypto-operations, but provide secure write-only memory for private credentials, which is unavailable on many platforms.


## 8.1 A Crypto-subsystem in RIOT

We now introduce the design and implementation of a crypto-subsystem that integrates hardware with software components and allows for a fair comparison across multiple platforms and libraries. We base our implementation on RIOT [34], an open-source operating system for low-end IoT microcontrollers.

We decided for RIOT because it runs on many architectures (from 8-bit over 16-bit to 32-bit processors), provides multi-threading with a scheduler supporting fixed priorities and preemption, power management [318], and a powerful hardware abstraction layer. Security protocols utilize cryptographic functions, which are currently implemented as software solutions at the system level [150].

Alternatively, the `package` system can be used to integrate external libraries. RIOT includes wolfCrypt [402], an embedded library for symmetric and asymmetric crypto, Cifra [73] which implements common building blocks for symmetric crypto, TinyCrypt [177] and micro-ecc

(uECC) [189], both particularly minimizing memory, and Relic [22], which contributes a comprehensive list of symmetric and asymmetric cryptographic schemes with particular support for many elliptic curves. As such, these third-party libraries are not implemented against any OS APIs, yet. Our design concept integrates these components in a generic fashion and extends to further hardware platforms and libraries in a straightforward manner.

### 8.1.1 Design Space

The operating system grants access to cryptographic hardware. A driver controls the device and implements an agnostic OS API. In five sample use cases of this chapter, vendors provide a library to access low-level operations. We now consider design aspects of how to integrate these and future cryptographic components.

#### 8.1.1.1 Vendor Driver Integration

Capabilities of vendor drivers vary widely. We argue for using these implementations, though, to take advantage of specific vendor knowledge, testing, and to allow for sustainable maintenance. The `package` subsystem in RIOT clones, builds, and links external repositories during firmware compilation. In this way, third-party software does not require maintenance within the OS and can easily be updated. We implement vendor libraries as RIOT packages and provide software wrappers to integrate external code into the subsystem. It is noteworthy, that vendor libraries do not always perform at maximum performance since they are commonly implemented in a generic way, as we will show in Section 8.7.

#### 8.1.1.2 Context Abstraction

Cryptographic functions operate on an internal state (context `struct`). It is allocated for each driver instance and depends on the exposed state by a vendor implementation which includes hardware specific elements internally. When facing the OS, a context `struct` must abstract vendor specifics and implement common OS interfaces. Hence, every driver defines a common context `struct`, containing vendor specifics and optional elements to facilitate the OS integration. Consequently, users of the API must not dereference the context `struct` since it changes with different backends. This design decision prevents parallel operation of different backends for the same function. We argue that this is in line with common IoT deployments for three reasons: (*i*) Single-core OSs are not optimized for parallel processing because real-world IoT firmware is tailored to a single application. Consequently, excessive parallelization of crypto-operations is not expected. (*ii*) Computational and memory resources are scarce on constrained IoT devices. Our context abstraction keeps complexity low. (*iii*) The performance of crypto-peripherals increases software solutions by one order of magnitude (see Section 8.4), thus, successive hardware operations already outperform software notably.

#### 8.1.1.3 State Handling

Applications of cryptographic primitives are manifold, including security protocols or pseudo-random number generators, and may require individual maintenance of a crypto-functions state. We enable external memory allocation and state handling in our approach. Software implementations operate on an allocated `struct` in RAM. Cryptographic processors may reduce operations to one at a time, provided they rely on a single hardware state. This inflicts to read, save, and restore the hardware state between operations to achieve state independence. Especially for external processors, this increases completion time and RAM requirements to replay and store hardware contexts. We implement an optional read–save–restore behavior for such devices and evaluate the overhead in Section 8.7

#### 8.1.1.4 Concurrent Access

Cryptographic processors need protection against concurrent access. Certain vendor drivers implement mutexes internally, while others require protection by the OS. It is noteworthy, that hardware devices require protection and not a single crypto-function, since processors are commonly single resources. We lock/unlock a mutex per device before and after every hardware access. Few microcontrollers provide hardware acceleration units with more than one crypto-peripheral, which can operate in parallel. In that case, each device must be protected separately against concurrency. Dual-accelerators promise throughput enhancements when crypto is heavily used. We implement a management instance to the driver which is requested internally before every crypto-operation and delivers the next free device.

#### 8.1.1.5 Low Power Management

Low energy consumption is a core requirement in the IoT. Active peripherals prevent devices from sleep and consume energy. It is an obvious design choice to keep active time of a crypto-device at minimum, hence, the OS integration of a driver should enable hardware only when used. This performs efficient on peripherals that turn on and off fast. Other devices (*e.g.,* external chips), however, require additional resources during initialization, which is inefficient when requested excessively. To deal with on/off patterns, we follow two different approaches. (*i*) Devices that turn on fast are powered only during operation. This aligns well with the concept of other peripheral drivers in RIOT. (*ii*) Devices with long wakeup sequences are not unconditionally set to sleep after usage, since concurrent applications might require crypto access. Instead, we implement a user counter that increments on device allocation and decrements on release. The device is turned off when the counter decrements to zero. Vendor drivers commonly operate synchronously, hence, our approach only affects preempted driver calls. On the downside, successive requests from a single context will not benefit and require a manual power switch which we analyze in Section 8.7.

### 8.1.2 Integration of Crypto Modules

RIOT currently supports 208 boards and 117 different microcontrollers, all of which exhibit varying crypto-hardware capabilities. To allow for the implementation and use of crypto-based applications without considering the current hardware setup upfront, we design (*i*) a hardware-agnostic API and (*ii*) the dynamic configuration of the crypto-subsystem. We use a feature model to represent crypto-hardware capabilities to the build system. Our approach selects and compiles hardware features where possible, and additionally provides an extended configuration interface to the user. This generic approach is capable of handling any hardware component, provided it is correctly modeled.

#### 8.1.2.1 Module Design

Our layered approach to interface with different crypto-backends introduces two types of APIs that are exposed to the user: (*i*) The *basic cryptographic API* provides direct access to low level functions, for example, a single AES block encryption. (*ii*) The *cryptographic mode API* grants access to operation modes of crypto primitives, for example, configuring AES in Cipher Block Chaining (CBC), or Electronic Code Book (ECB) mode. Different backends are modeled as modules, each of them is a translation unit that provides an implementation of one public API, which allows specific selection by the build system.

A backend module (*i.e.,* a driver) can support one of the three levels of crypto-acceleration in hardware: (*i*) Full hardware acceleration, (*ii*) partial hardware acceleration, and (*iii*) no hardware acceleration. The first level is given by peripheral-, or external devices that provide full hardware support for a cryptographic mode (*e.g.,* AES CBC). The second level occurs when hardware support is only available for basic cryptographic operations. In that case, the operation mode (*e.g.,* CBC) is performed in software and needs access to basic cryptographic primitives (*e.g.,* AES block encryption). The software component, however, is agnostic to the specific cipher or hash in use. The third level represents the case with missing hardware acceleration units. Our abstraction of the cryptographic API allows to switch backend implementations seamlessly.

#### 8.1.2.2 Feature Model

We use Kconfig [366] to select compiled modules during the build process. Kconfig allows to define symbols which specify dependencies and conditional default values. A user can interact with Kconfig using existing tools (*e.g., menuconfig*) to configure the values of a symbol. Each block in our crypto-stack is modeled as a Kconfig symbol. Hardware capabilities are represented as non-visible boolean Kconfig symbols that indicate the availability of crypto-hardware. Crypto APIs are implemented as boolean choice symbols, which allows for a transparent replacement of crypto-backends. Consequently, Kconfig provides a list of exchangeable modules, which are mutually exclusive. The default activation of a module is handled by Kconfig which selects the

Table 8.1: Overview of typical on- and off-chip IoT hardware with their crypto-acceleration features that we analyze.

| MCU/ Speed/ Board/ Library / Feature | **nRF52840** (@64 MHz) Nordic nRF52840dk CryptoCell | **EFM32**(PG12) (@40 MHz) Silicon Labs Pearl Gecko EMLIB | **MKW22D** (@48 MHz) Phytec IoT Kit 2 mmCAU | **ATECC608A** (I2C@400kbps) Adafruit ATECC608 CryptoAuthLib |
|---|---|---|---|---|
| TRNG | ✓ | ✓ | ✓ | ✓ |
| SHA-256 | ✓ | ✓ | ✓ | ✓ |
| HMAC-SHA256 | ✓ | ✘ | ✘ | ✓ |
| AES-128 | ECB, CTR, CBC,CCM(*) CMAC/CBC-MAC | ECB, CTR, CBC,CCM(*),CFB CBC-MAC, PCBC,GMAC,GCM | ✓ | ECB, GCM |
| AES-256 | ✓ | ✓ | ✘ | ✘ |
| ChaCha20/Poly1305 | ✓ | ✘ | ✘ | ✘ |
| ECC | secp160k/r1, secp192k/r1,secp224r/k1, **secp256**k/**r1**, secp384r1, secp521r1 Ed25519, Curve25519 | secp192r1, secp224r1, **secp256r1**, sect163k1, sect163r2, sect233k/r1, | ✘ | **secp256r1** (**P-256**) |
| ECDSA / ECDH | ✓ | ✘ | ✘ | ✓ |
| RSA | ✓ | ✘ | ✘ | ✘ |
| Secure memory | 128-bit | 5 x 256-bit | 256-bit | 16 Key Slots |

module combination, given the hardware capabilities of the underlying platform. A user can still manually overwrite the default selection.

## 8.2 Experimental Setup

### 8.2.1 Platform Overview

Table 8.1 summarizes the hardware and its features that we use in our evaluation. We omit legacy algorithms, and deploy our experiments on state-of-the-art (nRF52840 and EFM32) as well as older (MKW22D) microcontrollers with *peripheral* crypto-acceleration. nRF52840 and EFM32 are a good representation of the current generation of devices with advanced crypto-peripherals that were designed for flexible deployment in general use cases. Furthermore, we use the *external* crypto-chip (ATECC608A), connected via an I2C bus. ATECC608A represents a series of external security devices that are protected against side channel attacks.

All platforms provide a true random number generator (TRNG), of which all except the MKW22D comply with NIST standards (cf., [199] for background on embedded random number generation). The ATECC608A implements a crypto-secure pseudo-random number generator (CSPRNG), which is seeded from a true random source in hardware. EFM32 and MKW22D deploy software assisted HMAC SHA-256 that utilizes hardware hashing. nRF52840 and EFM32 support multiple cipher modes in hardware, contrasting MKW22D and ATECC608A, which need software assistance. ECC is available on all platforms but MKW22D.

The nRF52840 and EFM32 offer secured key registers for performing crypto-operations, and the MKW22D has a tamper protected register. The ATECC608A has 16 non-volatile, write-

only memory slots to store secret elements. Keys are generated and maintained on the external device to prevent unauthorized access and erase on tamper detection.

### 8.2.2 Measured Resources

We measure processing time, energy consumption, and memory overhead and repeat experiments 1000 times presenting averages, if not mentioned otherwise. To allow for a fair comparison, all software runs on RIOT version 2020.07.

**Processing time.** We evaluate the processing time with a logic analyzer that samples at 12 MS/s by toggling an I/O pin via direct register access on the test device. Using this setup the measurement overhead remains negligible.

**Energy consumption.** We connect our test platform to a regulated voltage supply (Siglent SPD3303C) and evaluate the current consumption of each operation using a digital sampling multimeter (Keithley DMM7510 7 1/2) at 1 MS/s. A measurement period is marked by toggling I/O pins. To bypass unrelated current flows, we connect our probes in series with the MCU and turn off unused hardware components (by hardware switches or in software). We measure the current of ATECC608A on the external chip, only, for better compatibility. In practice, the nRF52840 is used to operate the device, which might sleep during crypto-operations.

**Memory requirements.** We evaluate the memory consumption and consider compile- and runtime properties. Our compile time measurements show the overhead on top of a minimal RIOT build by analyzing the ELF file. We accumulate all linked objects that are associated with a crypto-implementation. Numbers are differentiated w.r.t. to RAM and ROM memory. Runtime memory requirements include the size of data structures and the maximum amount of stack memory that has been used during execution.

Table 8.2: Performance of SHA-256 on 64 Byte inputs implemented in different software on the nRF52840.

| Impl. | Processing Time | | | Memory | | | |
|---|---|---|---|---|---|---|---|
| | Init [$\mu s$] | Update [$\mu s$] | Final [$\mu s$] | Ctx [B] | Stack [B] | RAM [B] | ROM [kB] |
| Relic | 1.00 | 97.25 | 90.25 | 116 | 1000 | 110 | 1.3 |
| RIOT Core | 1.08 | 112.40 | 119.40 | 104 | 600 | 74 | 1.3 |
| wolfCrypt | 5.13 | 69.25 | 68.87 | 112 | 600 | 74 | 1.8 |
| TinyCrypt | 13.75 | 97.75 | 97.25 | 112 | 600 | 74 | 1.2 |
| Cifra | 12.62 | 85.75 | 113.40 | 104 | 616 | 74 | 1.2 |

Table 8.3: Performance of a single block AES-128 operation implemented in different software on the nRF52840. RAM is 38 Byte for all platforms.

| Impl. | Processing Time | | | Memory | | |
|---|---|---|---|---|---|---|
| | Init [$\mu s$] | Enc. [$\mu s$] | Dec. [$\mu s$] | Ctx [B] | Stack [B] | ROM [kB] |
| Relic | 3.00 | 57.42 | 88.03 | 577 | 1476 | 13.8 |
| RIOT Core | 3.25 | 38.17 | 71.17 | 20 | 508 | 4.7 |
| wolfCrypt | 0.67 | 51.50 | 86.42 | 284 | 780 | 11.7 |
| TinyCrypt | - | 225.70 | 659.90 | 176 | 668 | 2.6 |
| Cifra | | | | | | |
| unprotect. | 49.37 | 60.37 | 77.25 | 180 | 732 | 1.7 |
| protect. | 1617.10 | 6338.12 | 6353.87 | 180 | 732 | 1.9 |

## 8.3 The Impact of a Software Implementation

Table 8.2 and 8.3 compare the impact of different SHA-256 and AES-128 software implementations provided by commonly available crypto-libraries in RIOT that we ran on the nRF52840 platform. In both cases, we applied the cryptographic function to an input vector with the size of one internal block (*i.e.,* 64 Byte for SHA-256 and 16 Byte for AES-128).

Relic, TinyCrypt, and Cifra require 190–210 $\mu$s for an *init-update-final* sequence to process a SHA-256 digest. Cifra is faster during *update* on the price of longer finalization caused by an extra copy of the hash value. RIOT Core is 20 $\mu$s slower because it involves repeated modulo operations during state update (FIPS PUB 180-4). Furthermore, RIOT includes multiple endianness conversions to operate on 32-bit arithmetic. wolfCrypt provides a highly optimized implementation with an unrolled mixing loop, at higher memory consumption. Disabling this optimization increases the processing overhead of *update* and *final* to approximately 100 $\mu$s each, but it reduces ROM requirements by 500 Byte. Context sizes and stack usage are similar in all implementations, except for Relic that uses 400 Byte additional stack for a global array with initial hash state values.

The impact of different software implementations becomes more pronounced for AES-128. Initialization is fast in RIOT, wolfCrypt, and Relic taking at most about 3 $\mu$s to initiate state and the AES key length. TinyCrypt does not expose a separate API call for initializing but handles it internally. In contrast, Cifra contains the key schedule (FIPS PUB 197) already on initialization which takes up to 50 $\mu$s. wolfCrypt, Relic, and TinyCrypt provide a dedicated API to trigger AES key expansion, while RIOT handles the expansion on every en-/decrypt call. In Table 8.3 we include key expansion overhead in the en- and decrypt column, except for Cifra.

Encryption is by a factor of 1.5–3 faster than decryption for the additional key inversion during

decrypt [83]. RIOT provides the fastest implementation for a single block, followed by wolfCrypt and Relic. Their implementations base on pre-calculated look-up tables (T-tables) which is a speed optimization for 32-bit platforms. Cifra (w/o protection) and TinyCrypt implement the default algorithm based on a substitution table (S-box). Surprisingly, the S-box implementation in Cifra (w/o protection) scales similar to the T-table approach, whereas TinyCrypt operates 4–10 times slower. This is due to multiple copies between the internal and externally provided state, as well as explicit clearing of the internal memory. Lookup table implementations are vulnerable to side channel attacks [369, 28], especially cache attacks, why Cifra (w/ protection) provides countermeasures by default, that increase the runtime by a factor of 100.

Context sizes vary widely. RIOT allocates only 20 Byte for one context, of which 16 Byte represent the internal AES-128 state. Cifra and TinyCrypt keep the expanded key (multiple "round keys") in their context structure, which increases memory by up to 180 Byte. In that way, round keys do not have to be re-calculated when encrypting repeatedly on one AES context. In contrast, working on multiple contexts heavily increases RAM consumption. wolfCrypt and Relic follow a similar approach, though, both `structs` are not tailored to 128 Bit keys. wolfCrypt unconditionally allocates memory for 256 Bit keys and Relic additionally stores raw keys and initialization vectors that are used for cipher modes. The stack usage correlates to the context sizes. Memory overhead in ROM is more distinctive. wolfCrypt and Relic store complete T-tables (10 buffers of 1024 Byte), whereas RIOT only stores half of them and generates the remaining values on demand. TinyCrypt and Cifra attain the smallest ROM footprint based on minimized lookup tables.

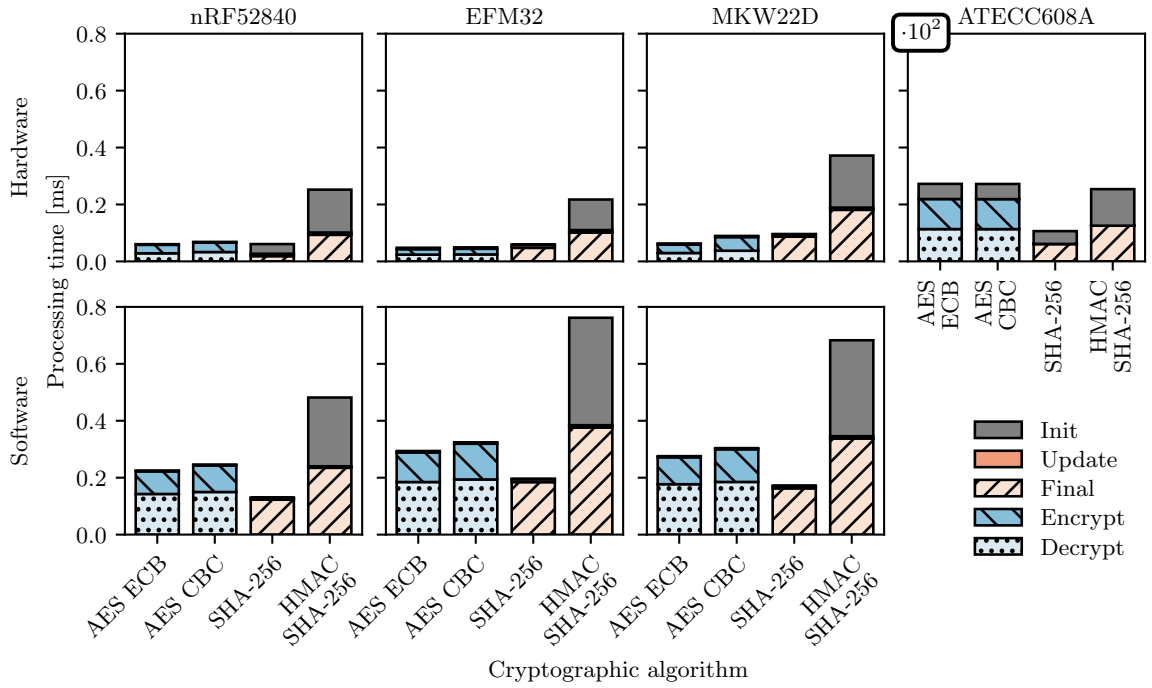
## 8.4 Basic Crypto-hardware Acceleration

Next we compare the performance of basic cryptographic operations between hardware and software, using the crypto-hardware discussed in Section 8.2.1. Software results are obtained from RIOT core on the same platforms but with crypto-hardware turned off.

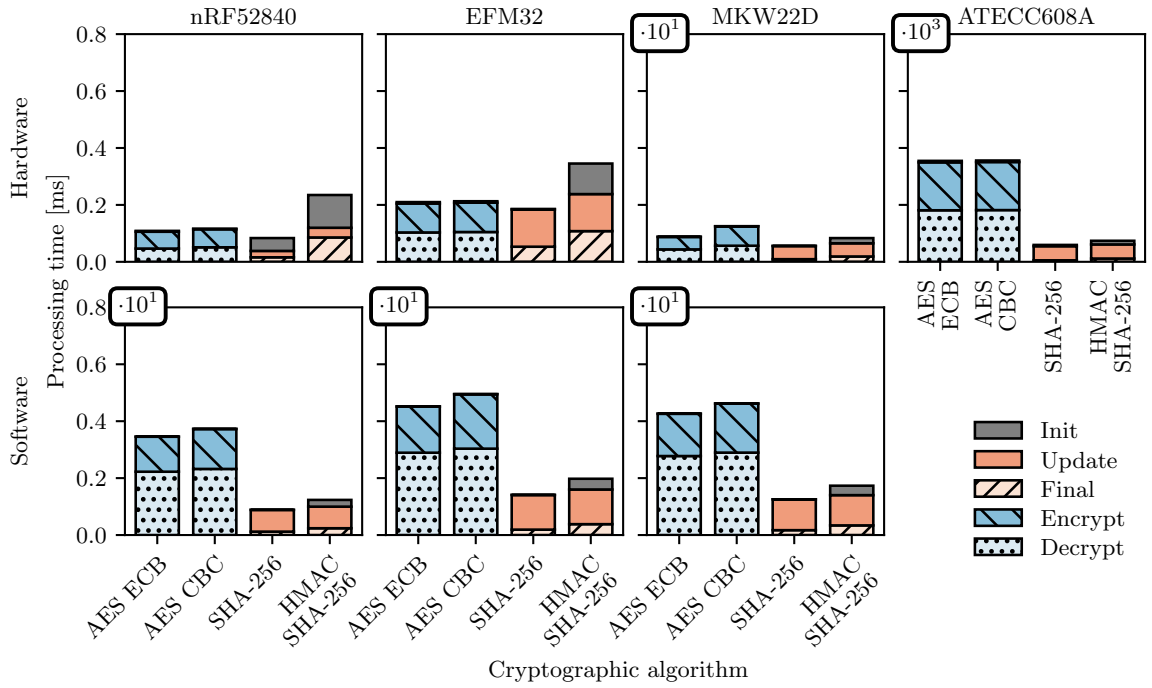
### 8.4.1 Processing Time

Figure 8.1 shows the processing time for short (32 Byte) and long (512 Byte) input data, separated into cryptographic operations. We use a randomly chosen 128 Bit AES key and a random initialization vector for the CBC mode. The HMAC SHA-256 is initialized with a random 256 Bit key. As occasionally recommended, we also conducted experiments with 512 Bit keys, but without further insights.

**Short input data.** Hardware accelerated operations scale similar on nRF52840 and EFM32 for short inputs (Fig. 8.1a) and require less than 70 $\mu$s for AES ECB/CBC during initialization, encryption, and decryption as well as SHA-256 hashing. HMAC SHA-256 is more complex, for repeated internal hash computations, and takes at most 250 $\mu$s on both platforms. Hash




(a) 32 Byte input data




(b) 512 Byte input data

Figure 8.1: Processing time of different crypto-algorithms on different platforms for short (32 Byte) and long (512 Byte) input data, separated into cryptographic operations. Crypto-operations are either accelerated in hardware on off-the-shelf microcontrollers (nRF52840, EFM32, MKW22D) or on an external cryptographic chip (ATECC608A connected via nRF52840), or purely implemented in software.

*updates* are small for all configurations due to the short input sequence. The *update* function collects 64 Byte of data (SHA-256 internal state) before starting a block operation, which is first triggered by *final* in this case. The MKW22D operates at minimal overhead for all functions. AES CBC encryption takes longer than decryption on that platform because of the software chaining of hardware accelerated AES blocks. An additional copy of the input buffer during encryption avoids overwriting it, which is omitted during decrypt.

When implemented in hardware, ciphers gain more—a factor 4–6—over software than hashes (factor 2–4). A comparison of software and hardware measurements for the EFM32 shows the particular power of that platform. It operates at minimal cost using hardware accelerated operations, in contrast to software, for which it performs slower than nRF52840 and MKW22D, since it operates at lowest CPU frequency. In software, AES ECB/CBC decryption is two times slower than encryption due to the additional key inversion (see Section 8.3). This overhead disappears on hardware.

The ATECC608A operates two orders of magnitude slower than the other platforms (see Figure 8.1a). The reason for this overhead is twofold. First, the vendor library maintains device power levels and wakes up the device prior to every operation. Second, control commands and data need to traverse the I2C bus with a copy to resp. from the microcontroller. AES initialization takes proportionally longer because the encryption and decryption key has to be sent to the device before use. The difference between cipher and hash based algorithms is higher on the ATECC608A in comparison to crypto-peripheral and software support, because AES-128 encryption of 32 Byte involves two block operations, the transport of which adds an overhead.

**Long input data.** Hardware crypto performance gains over software with longer input strings. We display results for 512 Byte input in Figure 8.1b. Hardware based hash computation now operates 5–10 times faster than software and ciphers speed up by a factor of 20 to 30. The processing time of crypto-peripherals still operates on a similar scale compared to short inputs, on nR52840 and EFM32. Surprisingly, accelerated operations on nRF52840 outperform EFM32 for long inputs—in contrast to short inputs.

The EFM32 requires a manual iteration over blocks, which involves a copy of intermediate data to a temporary buffer. The nRF52840 vendor library hides this iteration under its API and the internal operations are closed source. We expect advantages for nRF52840 here. Hardware accelerated AES on the MKW22D increases by one order of magnitude in comparison to short inputs, for the hybrid software-hardware chaining mode. The performance overhead for hashes is less dominant. Processing times for ciphers in software increase by one order of magnitude due to the complexity of block chaining and repeated key schedules. Hash computations are more comparable in software. The effort of *update* becomes visible since long input buffers update the internal state before calculating a final digest.

Most notable is a severe performance overhead on the ATECC608A, which operates three orders of magnitude slower than peripheral accelerators, and two orders slower than software.

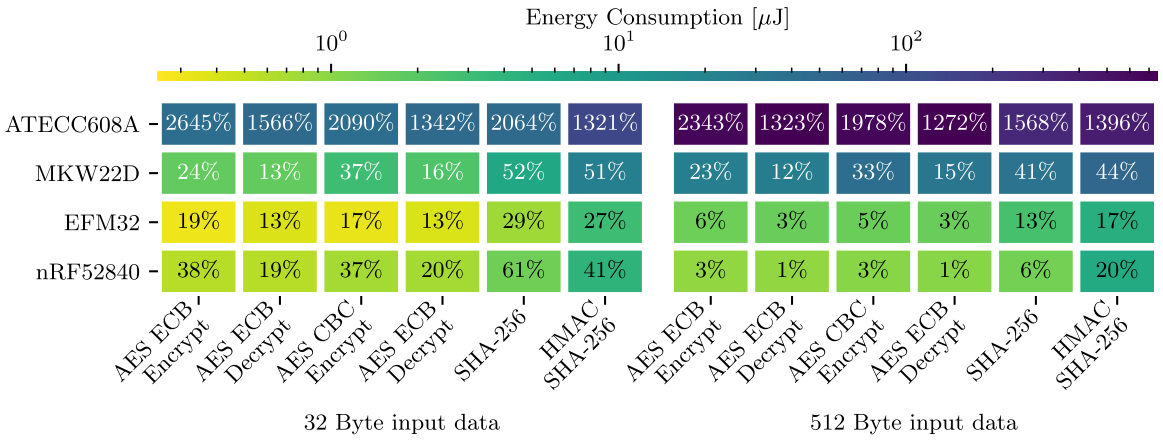



Figure 8.2: Energy consumption of hardware accelerated cryptographic operations on different platforms. Percentages show relative overhead compared to crypto-implementations in software running on the same devices.

AES suffers from the length of output data, which equals the input length. In contrast, HMAC- and SHA-256 only return a 32 Byte digest which relieves the bus and device control overhead.

### 8.4.2 Energy Consumption

Figure 8.2 depicts the absolute energy consumed by basic crypto-operations on hardware (colors) and displays the relative energy compared to software (percentages). AES encryption and decryption include the initialization, and hash based operations include *init-update-final*. These results roughly correlate with the processing time.

**Short input data.** The peripheral EFM32 and nRF52840 consume 0.25–4 µJ. EFM32 requires 4 mA and nRF52840 6 mA. The MKW22D is the most expensive peripheral and consumes 1–20 µJ due to a high current of up to 20 mA peak. The external ATECC608A consumes 45–130 µJ which is the highest energy demand despite its small average current of 1.2 mA—a consequence of the long execution time. HMAC SHA-256 is the most expensive operation on all platforms.

**Long input data.** The energy consumption of EFM32 and nRF52840 increase marginally over short inputs. nRF52840 now outperforms EFM32 despite the higher current of ≈1 mA. The MKW22D increases the consumption by roughly 10x, which is the overhead of software assisted acceleration. The most expensive device is still the ATECC608A whose consumption also increases by roughly 10x due to additional device management and bus utilization. AES based operations are the most expensive operations due to the high amount of encrypted data transmitted.

**Software versus hardware.** The performance gain from hardware acceleration increases with longer input data. The EFM32 and nRF52840 reduce the consumption on long input data down to 1 % of the software variants. The MKW22D equally profits for short and long input data.

Table 8.4: Memory consumption of different crypto-algorithms on different platforms. Crypto-operations are hardware accelerated. The overhead shows more (↑) or less (↓) hardware resources required in software (RIOT Core).

| Platform | AES ECB | | | | AES CBC | | | | SHA-256 | | | | HMAC SHA-256 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Ctx [B] | Stack [B] | RAM [B] | ROM [B] | Ctx [B] | Stack [B] | RAM [B] | ROM [B] | Ctx [B] | Stack [B] | RAM [B] | ROM [B] | Ctx [B] | Stack [B] | RAM [B] | ROM [B] |
| nRF52840 | 96 | 600 | 6407 | 7107 | 96 | 632 | 6455 | 7263 | 240 | 744 | 6391 | 5691 | 376 | 880 | 6455 | 6699 |
| Overhead | ↑76 | ↑36 | ↑6348 | ↑2105 | ↑76 | ↑20 | ↑6348 | ↑2117 | ↑136 | ↑136 | ↑6348 | ↑4477 | ↑168 | ↓16 | ↑6284 | ↑5213 |
| EFM32 | 20 | 524 | 84 | 4208 | 20 | 556 | 132 | 4288 | 104 | 600 | 64 | 4220 | 208 | 704 | 160 | 4492 |
| Overhead | 0 | ↓40 | ↑16 | ↓804 | 0 | ↓56 | ↑16 | ↓878 | 0 | ↓8 | ↑16 | ↑2990 | 0 | ↓192 | ↓16 | ↑2990 |
| MKW22D | 20 | 540 | 196 | 2838 | 20 | 556 | 244 | 2976 | 104 | 628 | 372 | 1692 | 208 | 924 | 468 | 1964 |
| Overhead | 0 | ↓24 | ↑136 | ↓2152 | 0 | ↓56 | ↑136 | ↓2184 | 0 | ↑20 | ↑328 | ↑480 | 0 | ↑28 | ↑296 | ↑464 |
| ATECC608A | 56 | 676 | 138 | 4175 | 56 | 740 | 154 | 4361 | 136 | 780 | 90 | 4089 | 136 | 748 | 154 | 4249 |
| Overhead | ↑36 | ↑112 | ↑79 | ↓827 | ↑36 | ↑128 | ↑47 | ↓785 | ↑32 | ↑172 | ↑47 | ↑2875 | ↓72 | ↓148 | ↓17 | ↑2763 |

Energy demands of the ATECC608A extension are much higher than for software—an increase by factors from 13 to 25 with only a slight gain in efficiency for longer input data.

### 8.4.3 Memory Requirements

Table 8.4 shows the memory consumption for basic crypto-operations on our reference hardware and compares the results to an equivalent software implementation.

**Context.** On nRF52840, the overhead of context structures ranges from 76 to 168 Byte, which are introduced by a buffer to represent hardware state for internal use by the vendor driver. AES CBC introduces less overhead since cipher chaining requires additional memory in software which is absent in hardware. In contrast, the EFM32 and MKW22D platforms require the same context sizes as our reference software as they do not mirror the hardware state.

**Stack.** The stack sizes allocated with the crypto-hardware are similar to the base crypto-software. Most notably for HMAC SHA-256, several accelerators can reduce stack size, *e.g.,* by not requiring a second SHA-256 context.

**RAM and ROM.** For ciphers, ROM overheads mostly decrease for the EFM32 and MKW22D due to the absence of static lookup tables in software (T-tables, see Section 8.3). RAM sizes remain moderate on these platforms with increases originating from initialized variables of driver libraries.

In contrast, on nRF52840 ROM and RAM overheads are dominant. Its crypto-library maintains an internal hardware abstraction layer, basic synchronization primitives, and interrupt routines. This software representation adds 6.4 kByte to all operations and cannot be disabled, even if features are not required for basic operations.

**External device drivers.** The external ATCC608A device interacts only via I2C communica-

tion and operates differently via its drivers. For example, encryption keys have to be written to a key slot and indexed for use, instead of passing a pointer to an allocated key structure. RAM and ROM usage are fairly independent of the selected crypto-operation and merely reflects the abstracted message transfer and invocation of the different hardware functions.

## 8.5 ECC Hardware Acceleration

We present analyses of elliptic curve cryptography running on hardware and software implementations. Hardware performance measurements consider peripheral crypto-acceleration of the nRF52840 and the external crypto-chip ATECC608A. Software measurements include Relic, a feature-rich library, and uECC, a minimal highly optimized library, on the same device. Relic operates in default configuration, which uses a precomputation table for scalar multiplication to improve runtime performance. uECC is deployed with optimization level two, which is the default to achieve a balanced speed-size tradeoff. We operate on the NIST P-256 elliptic curve with 256 Bit sized keys that is supported by all hardware and software platforms and evaluate keypair generation, signature generation and signature verification (ECDSA), as well as generation of a shared secret, based on preceding key exchange between two parties (ECDH). Signatures are computed on a 32 Byte message digest and secrets are 32 Byte (256 Bit) in size. Keypair generation and signing rely on random numbers and hardware accelerators use a built-in TRNG. Our software measurements use a seeded SHA-256 based CSPRNG. We also configured both libraries to use a hardware generator, but the advantage remains negligible. As the results do not contribute to additional insights, we excluded these experiments.

### 8.5.1 Processing Time

Figure 8.3 presents the average and the min/max processing time of different elliptic curve operations. Results now scale from tens to hundreds of milliseconds.

**Hardware crypto support.** The nRF52840 crypto-peripheral performs best, analogously to the results of Section 8.4. All operations require $\approx 20$ ms, which is one order of magnitude below the software. The ATECC608A operates 2 to 4 times slower than the nRF52840 peripheral. Verification as well as secret generation still outperform software by a factor of 2–3. In contrast to basic crypto-operations (cf., Section 8.4), the external device reveals distinct performance benefits because reduced device access contributes to lower control overhead.

**Software versus hardware.** Relic creates a precomputation table during initialization. This step, which takes up to 140 ms, is not required on hardware, and it is absent in uECC because lookup tables are statically compiled. Key generation and signing benefit from the precomputation table when applying the COMBS method [240] optimization for multiplying a prime elliptic curve point by an integer. This reduces processing times to less than 100 ms and leads to performance results that are on par with hardware accelerated operation on ATECC608A.

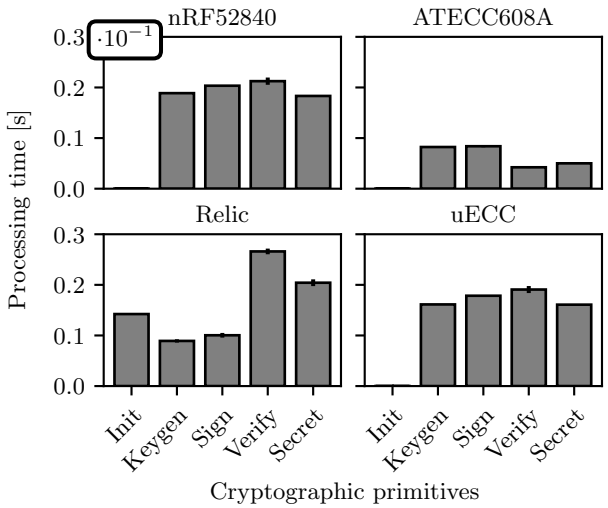



Figure 8.3: Processing time of different elliptic curve algorithms. Crypto-operations are either accelerated in hardware on the microcontroller (nRF52840), on an external chip (ATECC608A), or in software (Relic, uECC).

Verification, however, exhibits a higher processing demand of 260 ms. This variable base scalar multiplication is complex because Relic uses the window-non-adjacent form method [291]. The same applies to the shared secret generation. The ATECC608A, in contrast, performs inversely, which shows that a dedicated multiplication circuit is effective.

Compared to Relic, uECC operates at the same scale but benefits from selected algorithmic choices without using a precomputation table. Secret creation in uECC is 50 ms faster than in Relic because the co-Z Montgomery Ladder [273] outperforms the window-non-adjacent form for scalar multiplication over the elliptic curve [202]. Verification is equally fast in uECC taking advantage of the Strauss-Shamir method [356] for double scalar multiplication. Signature creation is 50 % slower, though. Considering the sum of signing and verification uECC and Relic are on par.

### 8.5.2 Energy Consumption

Figure 8.4 surveys the absolute energy consumption of hardware accelerated operations on elliptic curves (colors) and compares the relative excess over software (percentages). The ATECC608A consumes 700–1200 $\mu$J and the peripheral nRF52840 requires less than 400 $\mu$J for every operation. The most expensive operation on nRF52840 is signature verification (10 $\mu$J more than signing), which is in contrast to the ATECC608A, which requires more energy for signature creation but is most frugal when doing verification. These results roughly correlate with the processing time.

The performance gain of the ATECC608A over Relic surprises most. Both implementations

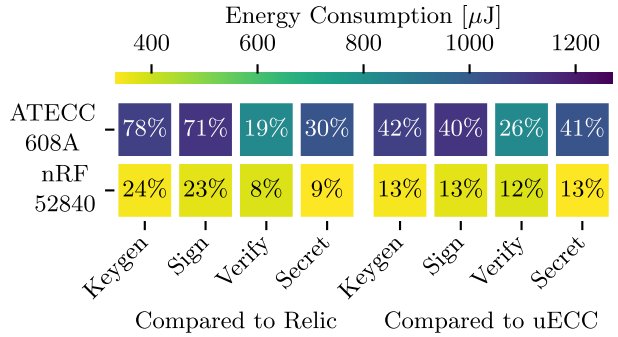



Figure 8.4: Energy consumption of different elliptic curve algorithms on the nRF52840 platform. Percentages show relative overhead compared to crypto-implementations in software running on the same device.

are on par in terms of processing overhead, but the specific hardware implementation requires only 71%-78% energy of corresponding operations implemented in pure software. Creating the precomputation table in Relic consumes approx. 500 $\mu$J and presents pure software overhead.

### 8.5.3 Memory Requirements

Table 8.5 shows the memory consumption for different ECC schemes implemented in hardware and in comparison to the two software libraries.

**Peripheral crypto support.** To store private and public keys the nRF52840 implements one data structure for each key, which require 816 and 884 Byte respectively. Considering private (32 Byte) and public (64 Byte) key sizes of the P-256 curve, this is a significant overhead and illustrates a design decision that favors flexibility at the expense of memory resources. Each data structure mirrors hardware state as well as the elliptic curve domain parameters (*i.e.,* elliptic curve modulus, equation parameters, and co-factors of a key). The latter contributes most to the overhead. Storing the domain parameters for each key (pairs) separately, however, allows not only different elliptic curve configurations in parallel but also to change parameters at runtime.

In contrast to basic crypto-operations (cf., Section 8.4.3), the stack of the nRF52840 operates ECCs at a scale of kilobytes because the crypto-driver library requires temporary buffers. A user context to sign and verify introduces additional overhead in ECDSA. RAM (8 kByte) and ROM (14–20 kByte) requirements of ECDSA/ECDH are large for off-the-shelf microcontrollers but relatively low (3 %) with respect to the overall memory available on the nRF52840.

**Software versus peripheral crypto support.** The nRF52840 crypto-hardware introduces the key `struct` and with it a significant overhead (520–820 Byte per key pair) compared to uECC and Relic. uECC requires only 64 Byte resp. 32 Byte for public and private keys. In Relic, the public key is represented as an elliptic curve point of 100 Byte, and the private key is contained in a multi-precision integer structure, which allocates 276 Byte per instance.

Table 8.5: Memory consumption of different elliptic curve algorithms on different platforms. Crypto-operations are hardware accelerated. The overhead shows more (↑) or less (↓) resources required in comparison to software.

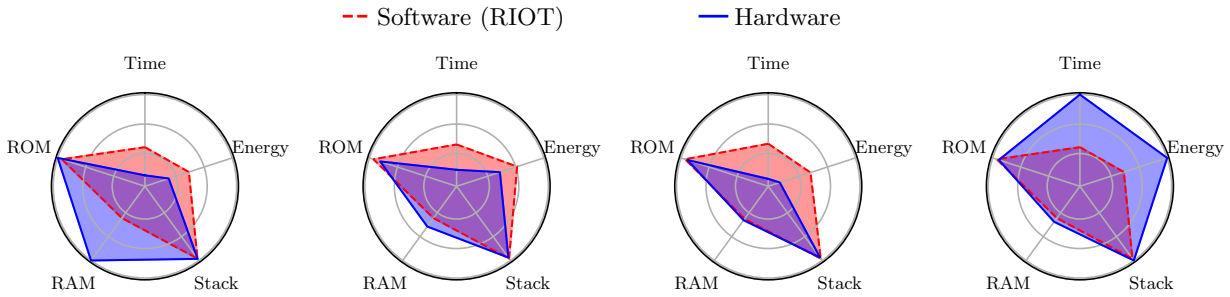
| Platform | Keys | | ECDSA | | | ECDH | | |
|---|---|---|---|---|---|---|---|---|
| | Private [B] | Public [B] | Stack [B] | RAM [kB] | ROM [kB] | Stack [B] | RAM [kB] | ROM [kB] |
| nRF52840 | 816 | 884 | 5472 | 8.08 | 21.20 | 3880 | 8.10 | 15.06 |
| Ovrh. uECC | ↑784 | ↑820 | ↑4352 | ↑7.70 | ↑14.28 | ↑3064 | ↑7.72 | ↑9.73 |
| Ovrh. Relic | ↑540 | ↑784 | ↓480 | ↑2.53 | ↓2.48 | ↓1136 | ↑2.55 | ↓6.22 |
| ATECC608A | – | 64 | 998 | 0.12 | 4.85 | 676 | 0.12 | 3.52 |
| Ovrh. uECC | ↓32 | 0 | ↓122 | ↓0.26 | ↓2.06 | ↓140 | ↓0.26 | ↓1.81 |
| Ovrh. Relic | ↓276 | ↓36 | ↓4954 | ↓5.43 | ↓18.82 | ↓4340 | ↓5.43 | ↓17.75 |

The nRF52840 requires more RAM, ROM, and stack than uECC. This is not surprising since uECC is tailored to a low memory footprint [347]. uECC is able to store key material and CSPRNG data in 380 Byte of RAM but provides only a limited set of features. Lower performance penalties are visible when comparing the nRF52840 with Relic. Similar to the nRF52840 crypto support, Relic is feature-rich and flexible. To implement this, Relic maintains a global context in the library, which leads to a similar performance overhead compared to the storage of domain parameters at the nRF52840. The nRF52840 still needs to run specific device drivers to control crypto-acceleration.

**External crypto-chip support.** The ATECC608A operates most frugal compared to the hardware crypto-peripheral and both software implementations. The low memory requirements enable its use even with tightly constrained microcontrollers. A core design advantage of this device is a dedicated, secure memory slot to store the private key. Only the public key data structure allocates a minimum of 64 Byte, while driver support impacts stack usage, RAM, and ROM only slightly. On the downside, ATECC608A limits crypto-operations to the P-256 curve.

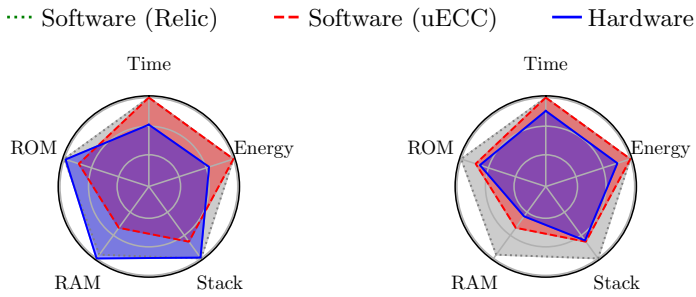
## 8.6 Comparison of Speed, Energy, and Memory

In this section, we compare the previously assessed performance metrics processing time, energy consumption, and memory requirements from a high level perspective. Figure 8.5 and Figure 8.6 display relative resource requirements of different cryptographic hardware and software implementations. For a better comparability, we scale the radius axis of each polar plot logarithmically, since the results span several orders of magnitude. Figure 8.5 shows the performance of AES CBC encryption plus decryption for 32 Byte short input data, acting as a representative




(a) nRF52840 (b) MKW22D (c) EFM32 (d) ATECC608A

Figure 8.5: Comparison of the processing time, energy consumption, and memory requirements of AES CBC encryption + decryption, for 32 Byte short inputs data. Crypto-operations are either accelerated in hardware on off-the-shelf microcontrollers (nRF52840, EFM32, MKW22D) or on an external cryptographic chip (ATECC608A connected via nRF52840), or purely implemented in software.




(a) nRF52840 (b) ATECC608A

Figure 8.6: Comparison of the processing time, energy consumption, and memory requirements of ECDSA signature generation + signature verification. Crypto-operations are either accelerated on the microcontroller (nRF52840) or on an external cryptographic chip (ATECC608A connected via nRF52840), or in software (Relic, uECC).

case for symmetric crypto-operation. Peripheral crypto-hardware (*i.e.,* nRF52840, MKW22D, EFM32) operates fast and energy-efficient when compared to software. In terms of memory footprint, the nRF52840 presents an exception, and notably increases the RAM requirements compared to all other hardware and software implementations, which the device is able to handle, though. External crypto-hardware (*i.e.,* ATECC608A) introduces a huge speed- and energy overhead for symmetric operation, but only a small memory footprint, enabling crypto for very constrained IoT devices [56] that do not provide enough resources for software operation.

Figure 8.6 shows the performance of ECDSA signature generation plus signature verification, acting as a representative case for elliptic curve cryptography. Our performance results for peripheral crypto-hardware on the nRF52840 reveal a similar picture compared to symmetric crypto-operation. It outperforms software (Relic and uECC) in processing time and energy consumption, but the nRF52840 driver again introduces RAM overhead. The external crypto-chip ATECC608A, however, reveals a different picture compared to symmetric operation and now outperforms software in terms of processing time, energy consumption, as well as memory footprint.

## 8.7 The Impact of Driver Implementations

### 8.7.1 Vendor Driver and Concurrent Access

The EFM32 (V. PG12) provides two crypto-peripherals that can be operated independently. This concurrent feature is managed to organize the peripheral access with a driver API that must be asynchronous in a single-core system. The vendor implementation, however, blocks the CPU during crypto-operation. Figure 8.7 (top) visualizes the progress of our test application using the vendor driver. We start two threads of the same priority, each of which encrypts data periodically. Thread 0 ($T_0$) triggers encryption on peripheral CRYPTO0. The CPU is acquired by $T_0$ until completion of the hardware. Thereafter, $T_1$ is scheduled and triggers encryption, operating CRYPTO 0 again since is not busy anymore. CRYPTO 1 is never used, since the vendor driver hinders parallelism.

We implement an asynchronous driver that exploits DMA to offload the CPU. Each crypto-device uses two DMA channels, one for copying input data to peripheral registers, and the other to return encrypted data. Figure 8.7 (bottom) visualizes the progress of our test application with an optimized driver. $T_0$ triggers encryption on CRYPTO 0 and relieves the CPU while the peripheral operates. When $T_1$ is scheduled, it triggers encryption on CRYPTO 1 and relieves CPU access. Note that both peripherals operate in parallel now, while the CPU stays idle. During that time, the OS can schedule other tasks, or switch to an energy-saving state. After completion of the peripheral tasks, each thread is notified.

Table 8.6 presents results of a test program that encrypts data concurrently, using AES ECB with and without optimization. Comparing the results for 32 Byte input data, our driver

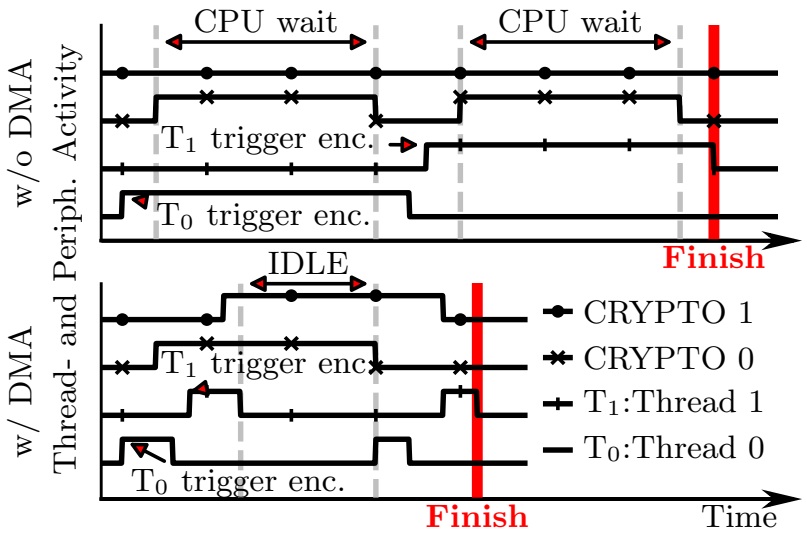



Figure 8.7: Qualitative comparison of thread and crypto-peripheral activity with (bottom) and without (top) CPU offloading using DMA.

Table 8.6: Processing time for 2000 AES-128 encryptions from two threads (1000 encryptions each) on the EFM32 with different diver implementations.

| | 32 Byte input | 512 Byte input |
|---|---|---|
| Implementation | Time [ms] | Time [ms] |
| DMA off, blocking | 37.10 | 205.20 |
| DMA on, non-blocking | 56.60 | 56.90 |

introduces an overhead of 20 ms for the DMA synchronization. For 512 Byte input data, the advantage gets visible. The vendor driver increases the progressing time by a factor of five, while our DMA fix remains unaffected and outperforms the vendor driver by 400 %.

### 8.7.2 Power Management and State Handling

Crypto-peripherals and external devices consume energy, why they should be disabled when not in use. Power-cycling a peripheral (*e.g.,* nRF52840) is fast, whereas external devices such as the ATECC608A have a longer, costly startup time. The ATECC608A vendor library turns off the device after every command, which is not desired on successive requests.

We implement a manual switch in the power management to prevent redundant power cycles. The ATECC608A requires to sleep in predefined intervals for clearing register and RAM values intermittently. This is enforced by a hardware watchdog timer, which can be configured to 10 s at most. Wakeup is triggered by a 100 $\mu$s low pulse on the I2C SDA line which can

Table 8.7: Performance of SHA-256 on the ATECC608A Platform with different driver properties.

| Implementation | Ctx [B] | I2C@100 kbps Rate [kB/s] | I2C@100 kbps Energy [$\mu$J] | I2C@400 kbps Rate [kB/s] | I2C@400 kbps Energy [$\mu$J] |
|---|---|---|---|---|---|
| Auto on/off | 136 | 1.73 | 85 | 3.01 | 48 |
| Copy state | 235 | 0.56 | 243 | 0.99 | 129 |
| Man. on/off | 136 | 2.70 | 59 | 6.46 | 30 |
| Copy state | 235 | 0.88 | 171 | 2.17 | 75 |

be generated by sending a 0-Byte when the bus is operated at 100 kbps. The ATECC608A, however, is capable of 400 kbps bus speed. To exploit the maximum performance, the OS needs reconfiguration capabilities for the I/Os to toggle the SDA pin independently of the I2C operation. We implemented this feature on the microcontroller that drives the ATECC608A. This implementation was used already in the previous sections.

The ATECC608A operates on a single hardware state (see Section 8.2.1) and requires an atomic *init-update-final* during hashing, though, certain use-cases operate on multiple hash states to be updated independently, before a final message digest is calculated. We implement an alternative SHA-256 function that replays the hardware state for every operation.

Table 8.7 shows evaluations for repeated SHA-256 operations on the ATECC608A. We measure the impact of manual power management, the I2C bus speed, and the overhead of preserving hardware state. Our evaluation presents context sizes, data rates for periodic hashing, and the energy consumption for a single hash. Manually powering the ATECC608A almost doubles the rate due to the reduced wakeup time. Conversely, the energy consumption reduces by a factor of 1.5. Increasing the I2C from 100 to 400 kbps increases the rate by a factor of two and decreases the energy consumption respectively. Copying the internal state, however, costs performance. The state needs to be stored in the context `struct`, which adds 99 Byte. The overhead of this mechanism reduces the rate by a factor of three and affects the energy consumption similarly. All together energy and speed can vary by on order of magnitude.

## 8.8 Related Work

**Crypto support in operating systems.** mbed OS [23] is the ARM operating system for the Cortex-M family. It includes the SSL library mbed TLS [24], which implements symmetric and asymmetric cryptographic algorithms in software. mbed OS allows replacement of cryptographic functions by crypto-hardware implementations. An analysis of the performance

advantages is missing, though. zephyr [412] does not include crypto-hardware, but support is planned for future releases[1]. Software-based crypto support is inherited from mbed TLS and TinyCrypt [177]. Similarly, mynewt [21] uses mbed TLS and TinyCrypt. A rudimentary crypto API provides `encrypt`/`decrypt` functions that leverage hardware capabilities for basic AES. Contiki-NG [20] provides sparse crypto support, though. With our crypto-extensions in RIOT we aim to bring more diversity in terms of hard- and software support to evaluate and deploy security in the IoT.

**Performance of crypto-software.** On very constrained devices, Gura *et al.* [157] compare RSA with ECC, Zhou *et al.* [416] present optimized implementations of SM2 and the NIST P-256 elliptic curve. These evaluations run bare metal based on software dedicated to specific hardware platforms which is in contrast to our study. We focus on a multi-purpose operating system and common crypto-libraries, and find that results are on par with bare metal implementations albeit our system favors more flexibility with respect to supported microcontroller platforms and peripherals. In the context of an operating system, processing overhead, memory, and energy were measured for symmetric cryptography (*i.e.,* AES) and secure hashing (*i.e.,* SHA and MD5) by Passing *et al.* [299] on NutOS and Tsao *et al.* [370] on Contiki [94].

Mössinger *et al.* [275] present runtime, memory, and energy consumption of elliptic curve cryptography in Contiki. Frimpong *et al.* [117] present an ECDH and ECDSA [9] implementations in Contiki-NG, using TinyCrypt. Kim *et al.* [203] ported the mbed TLS crypto-library to RIOT and FreeRTOS, and evaluate the processing time of ECDSA signature and verification on two platforms. Optimizations of elliptic curve cryptography are presented in [47, 244, 85]. In a comparative study of different elliptic curve libraries, Silde [347] shows that distinct optimizations for elliptic curves are vulnerable to side-channel attacks. This is one reason why we focus on common ECC.

**Performance of crypto-hardware.** Munoz *et al.* [276] present time and energy measurements for AES, running an SDK for software and hardware support on two platforms. Pearson *et al.* [302] compare the performance of peripheral and external crypto support for different symmetric and asymmetric operations, deploying Espressif and Arduino code, because a multi-platform OS was missing. Lachner *et al.* [220] assess time measurements for different ciphers and one asymmetric signature algorithm, operating three devices with Arduino firmware. wolfCrypt [402] includes crypto-hardware drivers and analyzed throughput of selected platforms. The library does not abstract hardware and thus cannot benefit from crypto-hardware on the OS level.

Gerez *et al.* [123] compare the power consumption of a TLS session using RSA and ECDHE between a Raspberry Pi and an IoT device with crypto-hardware. Mades *et al.* [247] compare the battery runtime of a TLS stack with and without hardware acceleration. Nofal *et al.* [285] analyse the TLS handshake and record layer and present the energy consumption of three

[1] https://docs.zephyrproject.org/latest/security/security-overview.html

elliptic curves and two RSA configurations on two hardware platforms, with and without crypto-acceleration. Schläpfer *et al.* [329] provide a brief performance comparison between new secure elements and DTLS, using mbed TLS as a software platform. Durand *et al.* [95] quantify the energy demands of OSCORE. Zhou *et al.* [415] argue for a reprogrammable FPGA approach to implement optimized cryptographic algorithms. Conti *et al.* [77] present a novel IoT platform architecture with AES acceleration and evaluate the benefit of hardware over software crypto. Their findings are on par with our study, however, the specific design contrasts our approach that focuses on off-the-shelf hardware and software implementations.

## 8.9 Conclusions

In this chapter, to the best of our knowledge, we presented the first comprehensive comparison of multiple symmetric and asymmetric cryptographic algorithms, implemented in hard- and software and consistently evaluated on multiple constrained common IoT devices. For a representative set of crypto-peripherals as well as an external security device, we showed detailed system benchmarks to reveal design tradeoffs when implementing secure crypto-hardware support on a multi-purpose operating system for constrained devices. Our results include: (*i*) *Crypto-peripherals outperform software in runtime and energy. The benefit increases with longer input lengths.* This contributes to node lifetime. On the downside, drivers introduce memory overhead. (*ii*) *Context sizes and stack utilization of crypto-hardware operate at a similar scale as crypto-software.* Device complexity unsurprisingly increases the overhead. (*iii*) *External crypto-devices are slow on symmetric crypto-operations, but their performance advances are notably on asymmetric crypto.* A small memory footprint enables cryptographic operations on very constrained devices. Furthermore, a collection of hardware based side-channel countermeasures provides additional resistance against attacks. On the downside, the I2C communication introduces an attack surface. (*iv*) *Special care is required with crypto-drivers. We found several vendor implementations with large optimization potentials.* Furthermore, different levels of hardware crypto support require a configurable environment with different layers of abstraction and software assistance. This is provided by the OS and contributes to code reusability and portability while exploiting hardware features. We hope that our results will help to prevent performance pitfalls in the future.

# Chapter 9

# Random Number Generation in the Low-end IoT


## Abstract

Random numbers are an essential input to many functions on the Internet of Things (IoT). Common use cases of randomness range from low-level packet transmission to advanced algorithms of artificial intelligence as well as security and trust, which heavily rely on unpredictable random sources. In the constrained IoT, though, unpredictable random sources are a challenging desire due to limited resources, deterministic real-time operations, and frequent lack of a user interface.

In this chapter, we revisit the generation of randomness from the perspective of an IoT operating system (OS) that needs to support general purpose or crypto-secure random numbers. We analyse the potential attack surface, derive common requirements, and discuss the potentials and shortcomings of current IoT OSs. A systematic evaluation of current IoT hardware components and popular software generators based on well-established test suits and on experiments for measuring performance give rise to a set of clear recommendations on how to build such a random subsystem and which generators to use.


## 9.1 The Impact of Random Input on IoT Security

The Internet of Things extends the distributed Internet system at the edge by a new, massive set of constrained devices. Secure communication in the IoT relies on cryptographic protocols of consistent design and proper practical instantiation, which includes provisioning of random numbers. Protocol security is commonly built on primitives that can be mathematically proven to meet security requirements. Such proves often rely on the hypothetical presence of a random oracle [45] that produces truly random input on request. More advanced, complex protocols are then constructed by securely composing these primitives. Canetti [65] was the first to introduce a definition of protocol security that is provably preserved under composition.

Secrets such as keys or intrinsic function values are spontaneously derived from random input. Security degrades whenever randomness is flawed. Extending the desired level of security to the

constrained IoT edge is a hard problem and random number generation on embedded devices must be seen as one of the key challenges in this context. Cryptographically secure random numbers require statistical robustness to withstand statistical attacks. Cryptanalytic attacks, however, encompass additional attack vectors to disclose random state.

Whenever an attacker can break in the current state of the random system, he should not be able to calculate back previous random values. This robustness property is known as *forward secrecy* and can be achieved by applying a non-invertible cryptographic function. Conversely, the attacker in possession of the current state should neither be able to predict future random output, which is known as *backward secrecy*. Establishing backward secrecy requires entropy for refreshing the generator state. These operations can be costly on a constrained device, and it is the objective of this work to identify feasible solutions of high quality standards. We discuss details of qualitative requirements in Section 9.2 and evaluate the trade-offs in Sections 9.6 and 9.8.

A variety of prominent attacks are based on vulnerabilities of random number generators in real-world systems and the literature provides a plethora of cryptographic analyses and attack scenarios [115, 286, 355, 187, 188, 92, 314, 159, 91, 183, 128, 346, 64, 232, 368]. In reality, there are even more attacks anticipated, many of which target at zero day exploits. Kelsey, Schneier *et al.* [187] criticize a lack of a widespread understanding of possible attacks against random number generators among system developers. Given the shortcomings of many random subsystems, it is worth considering the corresponding attack surface of the specific environment.

In the following, we present and contrast three common attack taxonomies [187, 115, 355] to clarify attacks on the random subsystem of IoT devices: (i) a cryptographic perspective, (ii) an embedded device perspective, and (iii) a systems perspective.

### 9.1.1 Cryptographic Taxonomy

An attack on a random number generator is an intrusive attempt of distinguishing between the produced sequence and truly random numbers. This distinction would open doors to predict future outputs or reproduce recent outputs that might have been used for generation of secrets in the past. The situation becomes even worse if an adversary manages to direct future numbers of a random sequence. Kelsey, Schneier *et al.* [187] enumerate three classes of analytical attacks:

**Direct Cryptanalytic Attacks** monitor PRNG outputs to gain knowledge about the system in order to distinguish between pseudo-random output and truly random bits.

**Input-Based Attacks** require access to PRNG inputs (seeds and initializations vectors) to inject known test sequences and perform further cryptoanalysis on random outputs.

**State Compromise Extension Attacks** base on previously compromised internal state of the generator and enables prediction or backtracking within the pseudo-random sequence.

These three principle attacks target PRNG core functions and may lead to broken security

schemes and protocols. In the IoT, vulnerable implementations do not only affect privacy concerns *etc.* but may also lead to actual physical damage because of IoT actuators.

### 9.1.2 Embedded Device Taxonomy

Generating random numbers on computers with a human-machine-interface has been studied extensively in the past [64, 91, 92, 159, 183, 187, 314]. These approaches can also be applied to interconnected embedded devices with our without user interfaces but are exposed to additional attack vectors. Francillon *et al.* [115] introduce two types of attackers for the case of wireless sensor nodes.

**Remote Attackers** mainly target at cryptanalytic and input-based attacks. Without accessing the node directly, an adversary tries to compromise or manipulate the generator state, *e.g.,* by monitoring and disturbing communication channels. In this particular example, wireless noise has been used to generate randomness. An adversary with access to the local wireless network must thus be considered a potential threat to perform an *external attack*.

**Invasive Attackers** gained read access to the internals of a generator and compromised its state at one point in time. This definition does not include write access, or code injection. Perfect state knowledge of a deterministic pseudo-random algorithm allows the adversary to predict future outputs and it can reproduce sequences that have been generated in the past, *i.e.,* for cryptographic key generation. Unless true random values are added to the generator state periodically, the system remains fully predictable. The update interval determines the maximum time that a generator remains vulnerable. Compromising the state by read access is also named an *internal attack*.

### 9.1.3 System-centric Taxonomy

In the IoT, a large number of constrained embedded devices inter-connect to each others and to the global Internet. On the one hand, broadening the networking capabilities increases the attack interface, especially with availability from the outside of a local network. On the other hand, simple IoT devices entail special considerations in comparison to traditional networked devices such as servers, personal computers, or smartphones. Many IoT devices are very constrained in hardware capabilities due to minimizing price and form factors, as well as energy resources. These limitations do not only affect computational power, but often imply sparse hardware protection features. As a consequence, IoT devices often lack permission management for code execution, memory protection mechanisms, as well as secret storages.

IoT deployments can grant physical access to the hardware, which opens a potential interface to analyse and monitor delicate key material, firmware, or even program execution on a device if tamper detection is not in place. We argue that secure random number generation cannot be sustained in the case that an adversary has full read or write access to the device. Shielding attack vectors without read or write access demands for additional hardware capabilities and

manufacturers of low-power chips already reacted. STMicroelectronics [355] defines three groups of attacks against microcontrollers (MCUs).

**System Software Attacks** focus on security and resilience affected by weak implementations, bugs, or insecure protocols after analyzing or even manipulating program execution. Disturbances are possible even without device access via network interfaces, *e.g.,* by sending malicious packets, or by triggering the execution of non-verified or untrusted library functions that may be already part of the device firmware. The latter often relates to "monkey testing" or to insider knowledge.

**Hardware Non-invasive Attacks** require hardware access. This category includes any kind of interface that allows interacting with the device directly, such as debug ports, or bus interfaces (UART, SPI, I2C, ...). The most dangerous attacks for random sources that rely on physical processes are based on fault injection. Typically, an adversary exploits the device under environmental conditions that it is not designed for. Prevalent fault injection parameters are temperature variations, microwave induction or voltage manipulation. Furthermore, side channel analysis such as power profiling and timing analyses fall in this category.

**Hardware Invasive Attacks** cover advanced techniques that enable access to the device silicon with access to hidden secrets, even if device protection mechanisms are in place. Such attacks are usually very complex and require specific measurement instruments.

Recently, hardware manufacturers of low-power microcontrollers have started to provide different countermeasures to the physical attack surface, ranging from debug port locks, tamper detection indicators, memory protection units, and isolated code execution environments. Also, hardware crypto acceleration on the chip becomes more widely available. These features should be used wherever possible. Nevertheless, many low-cost devices that are supported by IoT operating systems do not provide all (or any) of these capabilities. Implementations and algorithms used for random number generation should be designed around the concepts of (i) a high modularity to ease partial use of hardware protection features and (ii) robustness even if hardware protection is missing.

Random number generation cannot be protected, if the adversary has full control over the device. Analytic attacks as well as fault injections can be shielded, though, by incorporating cryptographic primitives and carefully gathering entropy for seeding the PRNG, as we will discuss next.

## 9.2 Generating Randomness in the IoT

Every day in the life of an IoT device, random numbers are requested by a variety of use cases. These use cases separate into two classes: either *general purpose* or *cryptographically secure* random input. While general purpose use only requires sufficiently well represented statistical properties, cryptographically secure random numbers must also remain unpredictable even under

malicious attacks. While the first category can be achieved fairly easily, the provisioning of secure randomness is very challenging in the constrained regime and—depending on the attacker model and strength—may not be achievable at all.

### 9.2.1 General Purpose PRNGs

General purpose PRNGs are employed for tasks independent of security aspects. Typical use cases include the jittering of network protocol timers or media access protocols (*e.g.,* random back-off in CSMA) to avoid collisions on a medium. Other applications of general purpose PRNGs include randomized sampling of sensor measurements and fuzzy testing.

A uniformly distributed stream of statistically independent random numbers is the desired output of a PRNG, which still should be approximated in higher dimensions, since concurrent applications or algorithmic elements may call on sub-sequences of the generator (*e.g.,* access every k-th output). Seeds between otherwise identical devices must differ to avoid identical random behavior across devices, and individual seeding after each device restart is desired. Even though seed requirements for general purposes are moderate, a "plug and play" source for gathering seed material is missing on IoT devices that do not provide a hardware based true random number generator.

General purpose PRNGs are essential on most IoT devices and frequently called in many use cases. Implementations should therefore be fast and efficient to preserve resources of the constrained devices. Available resources are better spent on generators with high security demands.

### 9.2.2 Cryptographically Secure PRNGs

Crypto-purpose or cryptographically secure PRNGs (CSPRNGs) are generators that are safe to use in security applications, involving the generation of cryptographic keys, nonces, or salts. Shamir [336] introduced the notion of a cryptographically strong PRNG which prevents computation of a desired future output value within certain bounds of time and space complexity. Blum *et al.* [53, 54] introduced cryptographically secure pseudo-random sequences that can be generated in polynomial time, but are unpredictable. Given a preceding output sequence of that generator, but not the seed, it must be computationally infeasible to predict the next bit of output with a better chance than 50 %. Cryptographic system security relies on these random numbers as basic input. Consequently, CSPRNGs are expected to output highly unpredictable number sequences and to be resilient against known attacks. The security of an implementation goes beyond the scope of computational efforts to predict future outputs and includes countermeasures to protect against weak implementations as well as state compromise by an attacker.

Building a crypto-purpose generator is more complex and consumes more system resources than a general purpose PRNG. It involves additional building blocks of ciphers, cryptographic hash functions, runtime tests, as well a specifically robust seeding logic. Computational over-

head and especially memory requirements of these building blocks are in potential conflict with resource constraints of IoT nodes, but the availability of secure random numbers is essential for enabling secure communication over the Internet. In order to reduce software complexity, some microcontrollers provide hardware acceleration of cryptographic primitives, which should be exploited when implementing the respective components.

A significant body of work reports about failures of PRNGs and successful attacks against the random input of crypto systems [187, 115, 92, 314, 91, 183, 355]. Hence, provisioning a cryptographically secure, consistent random infrastructure is a crucial component of a software system, which should be designed and tested with care. Requirements on CSPRNGs, in-depth analyses of different mechanisms and classification of those have been presented in [201, 43, 40, 351, 314, 92, 81]. We summarize the key aspects of CSPRNGs in the following paragraphs.

**Statistical Randomness.** Any statistical bias gives rise to elementary attack vectors. Even though CSPRNGs mainly consist of deterministic algorithms, a crypto-secure random generator needs to produce sequences that are statistically indistinguishable from truly random [187]. These properties base on the assumption that in a string of (pseudo-random) bits, probabilities for *one* and *zero* are equal at any time, and they are statistically independent. Even a very small bias must be considered as potential breach of the randomness assumption and contradicts crypto-requirements of a secure generator. A variety of statistical properties can be verified with tests that are available in well established test suites (see Section 9.4).

**Unpredictability.** CSPRNGs require resistance against external and internal attacks (see Section 9.1). A common distinction exists between prediction resistance and backtracking resistance. In more detail, prediction resistance means that an attacker cannot guess future results in computational time by monitoring the generator history, even if the algorithm is perfectly known. To achieve this at a given statistical quality, the seed needs to be fully unpredictable. Furthermore, a crypto-secure PRNGs needs to be built on cryptographic functions, usually one-way hash functions and block ciphers that are practically not invertible and do not produce colliding output from different inputs.

Every cryptographic system needs to be designed according to a specified security level. An established threshold is 128 bit of secrecy [1, 313, 371]. Assuming an adversary had to guess a secret value, it would require trying about half the number of bit combinations, if all states are equally likely. For 128-bit secrecy this would be $2^{128-1}$ tries on average to brute force a collision, or $2^{128}$ in the worst case. This is currently considered secure for computational resources. Both the seed at the generator input, as well as the internals of its algorithm need to meet the expected security level. It is important to note that due to the *birthday paradox*, an attack on a cryptographic hash function can complete with a reduced number of tries [126, 44]. Furthermore, if an attacker gained knowledge of the internal state of a generator, it should be ensured that future output is only predictable for a very short time. According to NIST, this should be achieved by adding fresh and truly random values periodically to the internal generator state (see Section 9.2.3 on re-seeding).

Backtracking resistance protects against a reconstruction of previous values or even the seed after a state compromise. It implies that no correlation between seeds and generated output should be in place. This behavior is required to assure perfect forward secrecy within cryptographic protocols [166]. Backtracking resistance is realized by applying cryptographic functions to the internal state of the generator and hardened by storing state in protected memory, if available.

**High Entropy Seeding.** A truly random seed value is required to make the output of a CSPRNG unpredictable [40, 269]. Random bits are usually extracted from physical resources and the Shannon entropy [340] or the Minimum entropy serve as a measure of its randomness. In this context, physically random resources are often referred to as entropy sources. Physical sources of "randomness" typically exploit variations in electronic circuits (*e.g.,* clock drifts, uninitialized memory, analog-to-digital converter fluctuations), randomly noisy signals (*e.g.,* wireless noise, bit errors, thermal noise), or user input signals (*e.g.,* keystrokes, mouse clicks) which normally are unavailable in the IoT. These real-world entropy sources, however, do not always admit "full randomness" and additional compression methods are often needed for maximizing entropy. NIST [373] also advises that seed generation should not rely on a single entropy source, for resilience. The IRTF recently proposed methods for improving randomness obtained from weak entropy sources [81].

Full entropy seeds are required for secrecy in a CSPRNG, which is equivalent to requiring $2^{n-1}$ tries on average, for guessing a seed of length $n$ bits. If not seeded with sufficient entropy, an adversary may exploit internal state collisions and determine generator output faster. This can drastically degrade the security strength of the generator. Hence, great care must be taken to harvest the number of entropy bits that is required by the cryptographic strength of the system as defined by and compliant to the algorithm of the CSPRNG [403]. Caution is advised with implementations that limit the input length of the seeding function. Fresh entropy may be required repeatedly to re-seed the internal state of a CSPRNG in order to recover from a potential state compromise, or to serve multiple generator instances, as we will discuss in the next section.

**Health Testing.** The quality of cryptographic system components need particular attention, as it may degrade not only due to software bugs, but also due to hardware aging or side channel attacks. Most entropy sources used for (re-)seeding rely on physical processes and particularly benefit from testing. Self-testing demands increase when physical device access is possible. A variety of tests have been proposed by NIST, which should be executed on all functions of a PRNG and its seeder. These tests range from known answer testing during validation time up to health tests that are applied during runtime to monitor vitality of entropy sources as well as expected execution of deterministic algorithms. The focus of this contribution is not on testing, and we refer the reader to the specific NIST documents [40, 373, 41].

### 9.2.3 A Note on Re-seeding CSPRNGs

A CSPRNG can recover from potential state compromise by regular re-seeding [187, 201, 40, 403]. Re-seeding of PRNGs is often advised [158, 115, 109] but also under much debate in the literature [314, 91, 46]. Certain CSPRNGs proposed by NIST even build upon the concept of re-seeding [40].

Considering that a generator is perfectly secure and was seeded in agreement with its specified security level, while both its seed and its state are kept in full secrecy, then an adversary cannot predict the next output by guessing within computational time without any re-seeding. So re-seeding becomes unnecessary. In this ideal scenario, the only reason for re-seeding is to extend the finite period of the specific pseudo-random algorithm. The period of a generator describes the number of cycles it takes to run through all valid internal states. In practice, however, cryptographically secure PRNGs have long enough periods and are not affected by repeating pseudorandom output during their lifetime in a common IoT scenario. Re-seeding can even be disadvantageous as it introduces an interface to inject low entropy values to the internal state during runtime. This may foster state collisions and thus break the resilience against unpredictability.

Entropy is a fragile property that is (i) not always in place (ii) in many cases manipulable at physical device access and (iii) hard to estimate during runtime. Even worse, it may be hard to depict failures in case of a compromise. To avoid adding compromised entropy values during runtime, some common security procedures base on a "trust on first use" model [93], which contradicts the re-seeding approach.

Another vulnerability created by re-seeding was revealed by Ristenpart *et al.* [314]. Existing implementations of entropy collectors cache their outputs in memory pools because—due to its eventually long and indeterministic runtime—entropy gathering is often implemented as a parallel and asynchronous task. If these numbers are not consumed immediately, a memory without perfect secrecy exposes an attack vector. Thus, entropy pools and internal states should run in trusted execution environments, only. Especially on constrained IoT hardware, this is not always possible, which makes the case for entropy generation on demand. Conversely, the utilization of insecure memory technologies generally motivates re-seeding with fresh entropy values.

Entropy sources can get compromised during operation, but the opposite can happen, as well, if sources did not provide full entropy during PRNG instantiation. In that case, mixing additional entropy values to the internal state during operation can be rescuing. Finally, few crypto-forums argue that re-seeding protects in case of faulty implementations. Faulty implementations at hand, however, contradict many assumptions of a cryptographically secure system.

In summary, there are reasons in favor and against re-seeding of PRNGs, and we argue that a decision should be made by the designer in view of the underlying hardware capabilities,

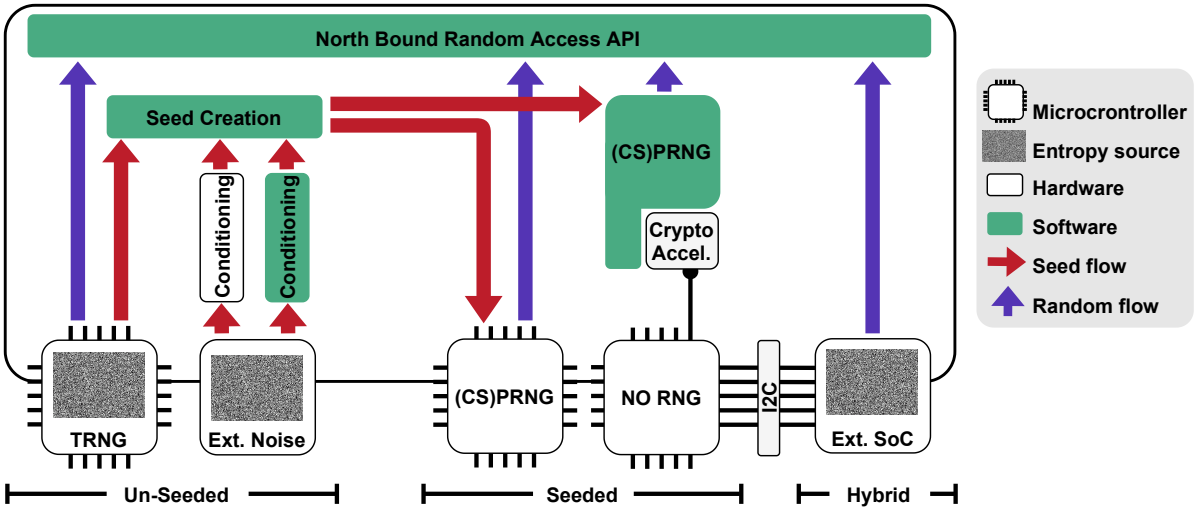



Figure 9.1: Overview of hardware and software components for generating randomness in the IoT.

deployment constraints, application scenarios, and security requirements in place. We conclude that a re-seeding mechanism should be considered as optional function of a CSPRNG API. This recommendation applies to IoT environments that require modular software in order to adapt to the heterogeneity of hardware platforms of varying resources and diverse deployments in probably (physically) harsh environments. These considerations, however, are not limited to resource-constrained embedded devices, but apply to regular computers as well.

### 9.2.4 System Components for Generating Randomness

Random numbers can be produced by hardware or software components. Combinations, in which assisting hardware improves functionality or performance of a software generator, likewise exist. Figure 9.1 presents an overview of different random sources that are commonly available on IoT devices. The access to the different sources is unified via a "North Bound Random Access API", which is commonly provided by the random subsystem of an OS. It is noteworthy, though, that these classifications also apply to ultra-constrained devices which cannot host an operating system. Such bare metal deployments may replace the random access API dedicated driver or PRNG calls.

We distinguish between (i) unseeded generators, (ii) seeded pseudo-random number generators, and (iii) hybrid solutions. (i) includes generators of truly random numbers. Many modern microcontrollers provide TRNGs that consist of an internal entropy source and a post-processing hardware circuit that compresses the samples from that source. These sources can feed into the random access API directly, even though it is debatable whether TRNGs should be deployed as an alternative to PRNGs (see Section 9.8). Alternatively, external noise from sources such as thermal noise of a resistor, jitter in free running oscillators, or uninitialized memory cells

gets sampled. As noisy data provides only few bits of entropy per sample, it needs a separate conditioning, which can be implemented in hardware [319] or software.

The output of TRNGs or (conditioned) noise sources is best used to create start values for seeded PRNGs. (ii) Deterministic general-purpose and crypto-secure PRNGs can be implemented in hardware on the microcontroller itself, or processed in software when random generating hardware is missing. Software PRNGs can additionally be assisted by hardware acceleration [196], which is available for crypto primitives on many platforms. (iii) Hybrid devices contain an entropy source and pseudo-random number generator in hardware, which utilizes the entropy for seeding. This class of devices is composed of dedicated crypto-chips that connect to the main processor using standard communication buses such as I2C, SPI, or UART.

## 9.3 Randomness in IoT Operating Systems

### 9.3.1 General Requirements

Many system services require access to random input, and it is common to expect a random function at the operating system level. Use cases and applications of random numbers differ largely, though, as we discussed in Section 9.2. General purpose PRNGs are needed to generate random events that follow a uniform statistical distribution and are often consumed at high frequencies. Security related contexts raise the additional requirement of keeping random output unpredictable, why crypto-secure generators need to maximize entropy with the help of truly random input. Such input is on the one hand difficult to obtain at often high cost, on the other hand truly random sources frequently harvest from system hardware, which is best accessed via the hardware abstraction of an operating system.

Following this perspective, both general purpose and crypto-secure random number generation should be part of an operating system, but are at the same time only versatile if they meet the diverging requirements well. We argue that the different use cases and requirements of PRNGs and CSPRNGs demand for independent methods and APIs. Isolated random functions cannot only be specifically optimized, but also prevent side channel attacks against the CSPRNG via the general purpose PRNG. Furthermore, separate APIs force developers to decide for their individual use cases.

### 9.3.2 General Purpose PRNGs

Use cases for general purpose PRNGs require statistically well distributed random sequences. First, single applications should receive a different value out of the whole number range on each request to avoid repeating patterns. Second, multiple applications that request from a single PRNG instance should experience the same properties, even if they access only every k-th PRNG output. This requires a decent empirical distribution in higher dimensions, which is often a challenge. Security related applications should not use this generator class as it may be

too easy to predict. Furthermore, most go-to PRNGs are invertible which allows to reconstruct previous sequences. Seeds must be generated differently across devices to prevent a uniform collective behavior—a decent entropy is desired to provide varying sequences between system restarts. Seeds may be accessed via the hardware abstraction of the operating system, but should be configurable to ease debugging.

General purpose PRNGs should be applicable even on very constrained devices and act frugally while requested frequently. Efficiency metrics involve processing time, as well as energy and memory consumption. The latter can benefit from restricting state to a single PRNG instance. A central instantiation logic can be managed by the operating system.

### 9.3.3 Crypto-secure PRNGs

**Core Requirements.** Security related use cases require crypto-secure PRNGs for sovereign tasks such as cryptographic key generation. Delicate key material must be largely unpredictable. A CSPRNG is expected to produce sequences that are indistinguishable from truly random numbers, as discussed in Section 9.2.2. It is advisable to rely on approved CSPRNG mechanisms that have been verified by trusted authorities and an operating system or a public library can provide access to implementations that are tested within this environment. A CSPRNG internally consists of cryptographic functions [40] to achieve backtracking resistance and the maximum achievable prediction resistance (security strength) is typically given by that function, though, the strength of the whole generator should be specified by the designer of the approved algorithm. In order to assure a predefined security strength, a high quality seed must provide truly random data with a corresponding amount of entropy during instantiation of the CSPRNG.

In contrast to general purpose PRNGs, CSPRNGs undertake tasks like key generation, which is typically involved less frequently, but also continuous encryption within stream ciphers, thus, performance characteristics of CSPRNGs are important, but secondary in comparison to its security qualifications. Still, computation of cryptographic primitives and entropy conditioning can be costly [91], in particular on constrained embedded devices. The OS CSPRNG and its seed generator must comply with the constraints of the target hardware and leave sufficient resources to deploy a real-world firmware that includes a crypto-stack and the desired application logic. The operating system should support this in an optimized, configurable environment.

**Minimal Standards.** Crypto-secure PRNGs rely on true random seeds that meet a security strength which determines the required amount of entropy in the seed. At least one entropy source must be in place that meets the requirements. The operating system should provide an entropy interface, which grants access to true random values generated from varying sources, dependent on underlying hardware capabilities. Externally connected devices may provide true random numbers as well, but typically require a device driver. An operating system can simplify access to relevant components by its hardware abstraction layer, and should additionally allow for code re-use between different hardware platforms. Configurability requires a highly modular

architecture. In the context of IoT software, configuration is commonly done during compile time to keep firmware sizes small.

Tests are mandatory for generating robust and secure random numbers [40, 373]. Both the pseudo-random algorithm and the seeding entropy source must be tested, whereas testing procedures can be separated into a priori and live tests during deployment. Due to device constraints, a priori tests at development time should be favored to save resources on running IoT nodes. Thereby, bug free execution of the deterministic CSPRNG must be verified by comparing output sequences against ground truth. Further, seed sources rely on physical processes and should be evaluated within deployment conditions, because their behavior can be affected by environmental properties.

**Optional Features.** Entropy sources are essential for seed creation, but the properties of underlying physical processes are diverse. Environmental changes as well as attackers with device access can affect their reliability (see Section 9.1). Involving multiple entropy sources during seed creation increases seed resilience. Naturally, a physical process does not provide full entropy, but conditioning is sometimes implemented already in hardware on the microcontroller. For sources with sparse entropy concentration, a compression mechanism is required. An entropy module provided by the operating system can increase seed quality, and it should involve three fundamental building blocks: (i) An estimate about the entropy amount per input which can be provided by each source, (ii) an accumulation instance to involve multiple sources and keeps track of the amount of accumulated entropy, and (iii) a compression mechanism to create high entropy seeds of limited length to meet security requirement of the CSPRNG. Steps (ii) and (iii) can be combined in one function. The entropy API should provide an interface to pass security requirements, and it should also be able to report errors back to the CSPRNG.

Re-seeding a CSPRNG is sometimes desired to recover from potential state compromise (see Section 9.2.3). We argue that re-seeding should be kept optional because seed generation may drain a significant amount of energy on every re-seed cycle. Enabling and disabling that feature should be transparent to the application that uses the CSPRNG.

Cryptographic protocols for different purposes require to operate on individual instantiations of a CSPRNG. The internal state is a vulnerable asset and facilitates forward- and backward tracking. Hence, it should be isolated, to prevent tampering through backdoors as well as side channel information leakage. This affects the operating system in two ways. First, seeding needs to be done separately for each instance—in contrast to the unified approach suitable for general purpose PRNGs. Second, every instantiation needs its own context to be handled either internally within the boundaries of the CSPRNG or externally, by dedicating context allocation to the application. In both ways, the number of contexts should be kept low in an IoT OS, because the CSPRNG state can consume much memory. It is worth noting that IoT firmware typically avoids dynamic memory allocation, why the number of CSPRNG instantiations should be explicitly defined during development.

The state of a generator needs protection and an operating system should involve hardware

security features, if available. In more detail, secure memory technologies can provide tamper detection along with authorized access and recent low-power platforms even provide trusted executions environments (*e.g.,* ARM TrustZone [25]) for protected code execution. CSPRNG state, seeds, and entropy values should be uninstantiated after use to avoid leakage via side channels. This is of particular importance when secure memory is absent. Memory erasure is commonly done by setting buffers to zero, though, instructions to "zeroize" a buffer are often removed by compiler optimizations, which leaves sensitive information in memory. Known solutions involve *explicit_bzero* implemented by the GNU C Library, as well as service functions in cryptographic libraries such as libsodium [239] or Monocypher [271].

**Optimizing Parameters.** Modern off-the-shelf IoT devices provide on-chip TRNG hardware, which is commonly used for seeding. In addition, accelerating hardware units are often in place and capable of processing cryptographic primitives such as ciphers and hashes. While most CSPRNG implementations base on a pure software solution of their internal cryptographic functions, a transparent substitution by hardware implementations promises performance enhancements in terms of speed and energy. The operating system can provide peripheral drivers to control hardware accelerators and a transparent reconfiguration that accounts for hardware capabilities. It should automatically select the most performant solution.

Certain IoT boards provide an external hardware accelerator mounted on the same PCB as the microcontroller, which is typically connected by a peripheral bus. Such devices can transparently act as (i) a cryptographic accelerator or (ii) an alternate CSPRNG, which off-load the main processor while enabling cryptographic applications on limited devices that cannot run CSPRNG software.

### 9.3.4 IoT Operating Systems

Currently, the most prominent open source IoT operating systems are Contiki-NG, a successor of the original Contiki operating system [94], mbed OS [23], FreeRTOS [16], zephyr [412], Mynewt [21], and RIOT [34], all of which implement methods to gather random numbers. In the following, we give a brief overview about current solutions in different OSs and summarize the results in Table 9.1. We further focus on RIOT, which serves as the basis for our experiments.

**Contiki-NG** provides support for a few ARM Cortex-M based microcontrollers, and MSP430 platforms, although its predecessor Contiki provided support for a wider range of architectures. Contiki-NG implements a sparse random subsystem that does not distinguish between general purpose and cryptographically secure PRNGs. The random API is implemented as a wrapper around peripheral TRNG drivers of a platform. The random interface provides a seed function that limits the input size to an unsigned short integer, but it is left unimplemented in most cases, because the TRNG does not require a seed. In case of missing hardware random number support, the random module falls back to the C library function *rand*. An entropy module for seed generation is missing.

Table 9.1: Overview of common open-source IoT operating systems and their support for randomness.

| | Operating System | | | | | |
|---|---|---|---|---|---|---|
| | Contiki-NG | mbed OS | FreeRTOS | Zephyr | Mynewt | RIOT |
| **PRNG** | | | | | | |
| General-purpose | *rand*[a] (C Library) | Xoroshiro128+ | *rand*[a] (C Library) | Xoroshiro128+ | *rand*[a] (C Library) | Mers. Twist.<br>Tiny Mers. Twist.<br>Xorshift(32)<br>Park-Miller LCG<br>Knuth LCG |
| Crypto-purpose | ✘ | HMAC DRBG<br>CTR DRBG | HMAC DRBG<br>CTR DRBG<br>Hash DRBG | HMAC DRBG<br>CTR DRBG | HMAC DRBG<br>CTR DRBG | HMAC DRBG<br>CTR DRBG<br>Hash DRBG<br>SHA256PRNG<br>Fortuna |
| **Entropy** | | | | | | |
| Sources | TRNG | TRNG | TRNG<br>Timer[b] | TRNG<br>Timer/<br>Counter[b] | TRNG | TRNG<br>SRAM PUF<br>CPUID[b] |
| Accumul. | ✘ | ✘ | ✓ | ✘ | ✘ | (✓) |
| **Additional features** | | | | | | |
| Ext. state | ✘ | ✓ | ✓[c] | ✓[c] | ✓[c] | ✓[c] |
| Err. interf. | ✘ | ✓ | ✓[c] | ✓[c] | ✓[c] | ✓[c] |
| HW accel. | ✘ | ✓ | ✘ | ✘ | ✘ | ✘ |

[a] *rand* maps to an LCG–type PRNG in most libraries.
[b] Predictable source that varies output between calls or devices.
[c] Function or state is not exposed via an OS API.

**mbed OS** is the ARM operating system for processors of its Cortex-M family. mbed implements one general purpose PRNG with an API interface to re-seed the internal state. ARM maintains the SSL library mbed TLS [24] next to the operating system, which generates secure random numbers. mbed TLS is portable and used in other OSs. The CSPRNG implementations include external state handling, re-seeding procedures, and a collection of self tests. mbed TLS implements a dedicated module for entropy gathering, which (i) is capable of accumulating multiple entropy sources, (ii) provides an interface to add personal entropy sources, (iii) can be used in a blocking and non-blocking fashion until certain entropy requirements are met, (iv) distinguishes between weak and strong entropy requirements, and (v) can be compiled with different complexity levels.

**FreeRTOS** is a microcontroller OS which is positioned to meet hard real-time requirements. FreeRTOS provides support for a wide range of platforms, including ARM Cortex-A/M devices,

RISC-V, and MSP430. The operating system uses the C library function *rand()* for general-purpose random numbers. The external mbed TLS and wolfCrypt [402] security libraries are ported to FreeRTOS, which enable three different crypto-purpose generators. A common random API is missing on the OS level. Self tests, as well as health tests are inherited from the respective library.

**Zephyr** supports a large variety of ARM Cortex-M based 32-bit IoT platforms as well as x86, ESP32, ARC, NIOS II and RISC-V based boards. The operating system implements a PRNG for general purposes and the external mbed TLS and TinyCrypt [177] libraries are ported to zephyr, which include crypto-purpose generators. A collection of tests inherited from mbed TLS and TinyCrypt can be executed on the operating system, as well as selected benchmarks.

**Mynewt** is an operating system that supports about 40 boards, most of which with an ARM Cortex-M 32-bit microprocessor, but some also involve MIPS or RISC-V architectures. Similar to Contiki-NG, it provides access to a TRNG or general purpose random numbers via the C library function *rand()*. Both mbed TLS and TinyCrypt are available for cryptographically secure random number generation, but they are not accessible through an OS level random API. Selftests from mbed TLS can be executed within the operating system, whereas TinyCrypt tests are not included.

**RIOT** currently supports more than 200 boards involving 30 different microcontroller families that range from 8-bit AVR devices with sparse peripherals over 16-bit MSP430 devices to 32-bit processors of the ARM Cortex-M family or ESP32 with various MCU peripherals including dedicated randomness generation hardware circuits, as well as ARM7, MIPS, and RISC-V based microcontrollers. RIOT implements a collection of PRNGs including ultra-lightweight general purpose algorithms, as well as crypto-secure generators. Among other embedded crypto libraries, wolfCrypt, TinyCrypt, and relic [22] run on RIOT, but external CSPRNGs are not yet integrated into the random subsystem. A *random* API unifies access to pseudo-random numbers, but does not differentiate between general purpose and crypto-secure PRNGs. A test application with user interaction via the shell allows to evaluate vitality and basic performance metrics of a generator.

## 9.4 Statistical Test Suites for Random Numbers

The statistical quality of random sequences can be empirically analyzed with many methods and tools that assess random properties. The NIST Statistical Test Suite (STS) [43] and the DIEHARDER Random Number Test Suite [61] combine series of such tests. They are established as standard tools and openly available. Both suites base on hypothesis tests that analyse the input against the null hypothesis of perfect randomness. This hypothesis implies that fully deterministic pseudo-random sequences of ideal random properties cannot be distinguished from truly random values. Conversely, even ideal random sources may produce sequences that appear

to have non-random properties, which occasionally leads to failures of statistical tests—usually referred to as type-1 error.

### 9.4.1 NIST Statistical Test Suite

The NIST STS consists of 15 different test cases, some of which are executed repeatedly, leading to a number of 188 statistics that are processed on each run. With respect to the input size recommendation for each test [43], we apply the test suite version sts-2.1.2 to 100 Mbit generator output. Every test is repeated 100 times with 1 Mbit test sequences. A single test returns a probability value ($p$-value) and it is expected to accept the hypothesis of perfect randomness with a confidence of $1 - \alpha$, if the value lies above a significance level of $\alpha = 0.01$. Otherwise, the hypothesis of randomness is rejected and the result is interpreted as failure. Each test is applied repeatedly which results in a vector of $p$-values. The proportion of passed sequences for one test is determined using the *confidence interval*. As a next step, the distribution of $p-$values for each test is analyzed using a chi-squared test ($\chi^2$)-test, which outputs a second order probability value ($p_2$-value). The test suite defines a significance level of $\alpha_2 = 0.0001$ for testing this distribution.

### 9.4.2 DIEHARDER Random Number Test Suite

The DIEHARDER test suite subsumes 31 tests to analyse statistics of random input streams. These tests are executed with varying parameters, thus, a full run calculates a total of 122 test statistics. We apply the DIEHARDER test suite version 3.31.1 with default options to streams of raw binary outputs. In default mode, the number of repetitions of a specific test varies between *psamples* in [1, 1000] and the sequence length is variable between *tsamples* in [100, 65000000] which demands for much more random input data in comparison to the NIST STS. Repeated executions deliver multiple probability values ($p$-values) for each test, similar to the NIST tests. A Kolmogorov-Smirnov test (KS-test) is applied to test deviations from the expected distribution, which results in a second order probability value ($p_2$-value).

In DIEHARDER, a test passes if its $p_2$-value lies above a significance level of $\alpha_2 = 0.000001$, below $(1 - \alpha_2)$, and it fails otherwise. The result is considered as weak if $p_2$-value lies above $\alpha_w = 0.005$ and below $(1 - \alpha_w)$. It is again worth noting that even truly random numbers might generate weak results occasionally.

### 9.4.3 Other Test Suites

Donald E. Knuth was one of the pioneers who described randomness tests in early editions of *The Art of Computer Programming* [205] and his tests are part of most established test suites today.

NIST released the first random number tests in 1994 within the FIPS 140-1 [280] standard, which specified four statistical tests. In 2001, these tests were updated in FIPS 140-2 [281] with

a narrowing of the test criteria. Both documents served as predecessors for the NIST Statistical Test Suite (STS) released in 2010, which includes all 140 FIPS test cases as a subset. The Diehard test suite was published in 1995 by Marsaglia [253], who had been active in this field since years. The test suite implements a collection of 18 test cases, which are a central part of the DIEHARDER test environment, which has been developed since 2003. Both NIST STS and DIEHARDER are well known and accepted as standard tools for statistical testing of random number generators [225, 359].

The TestU01 library [225, 226] was introduced in 2007. It includes the majority of tests from NIST STS, Diehard, as well as additional tests proposed in literature. Its purpose is to provide an "extensive set of software tools for statistical testing of RNGs." [225], which led to a larger variety of tests, larger sample sizes and an extended test parametrization in comparison to the other suites. At the core, the environment implements hypothesis tests similar to NIST STS and DIEHARDER, but instead of rejecting a hypothesis, it simply reports $p$-values outside the interval [0.001, 0.9990]. TestU01 can be executed on four complexity levels, of which the most comprehensive one (*BigCrush*) involves up to 160 test statistics. Generation of the required amount of random data can take a long time, in particular when generated on microcontrollers and transmitted via the UART to feed the library. This drastically increases time requirements of the evaluation process.

A range of other test environments are less prominent in the literature. The SPRNG (Scalable Parallel Random Number Generators) [256] library is a tool to optimize distributed processing for parallel random number generation and it additionally contributes a few standard tests already covered by NIST STS and DIEHARDER. The ENT test program [388] defines a small-scale environment that executes only five statistical tests. It relies on a file based data input, which is not practical when huge datasets have to be analyzed. The CryptRndTest package [87] analyses cryptographic random numbers, focusing on high precision floating-point numbers with lengths larger than 64 bits. The latter is uncommon in the IoT.

## 9.5 Hardware Generated Random Numbers

Common off-the-shelf IoT platforms sometimes provide hardware generated random numbers. While some platforms implement "true random" circuits for entropy gathering on the same chip as the CPU, others implement pseudo-random generators in hardware. Still, many microcontrollers do not offer random generating hardware at all. In these cases, external components such as transceivers or cryptographic co-processors may be connected to a bus and contribute true random numbers. As an alternative, advanced mechanisms can extract random physical properties from manufacturing variations of the microcontroller itself. In this section, we analyse typical IoT hardware platforms from different manufacturers, CPU architectures, and feature sets. Results are summarized in Table 9.2. We run RIOT-2020.01 as operating system with a collection of custom measurement programs.

Table 9.2: Overview of the typical on- and off-chip IoT hardware with their random features that we analyze. (Abbreviations: RO=Ring Oscillator, LFSR=Linear Feedback Shift Register, RF=Radio Frequency).

| Board | Chip | Entropy Source | Post-processing | Error Handling |
|---|---|---|---|---|
| ST NUCLEO-F410RB | STM32F4 | 3 Free-running ROs | Bias Correction + LFSR | Health Tests |
| Phytec IoT Kit 2 | MKW22D | 2 Free-running ROs | LFSR | Status Indication |
| Nordic nRF52840 DK | nRF52840 | Thermal Noise | Bias Correction | Status Indication |
| Zolertia RE-Mote | CC2538 | RF Noise (seed only) | 16-Bit HWPRNG | – |
| Atmel SAM R21 XPRO | SAMD21 | – | – | – |
| Arduino Mega 2560 | MEGA2560 | – | – | – |
| Openlabs Radio Breakout | AT86RF233 | RF Noise | – | – |
| Microchip CryptoAuth XPRO-B | ATECC(5\|6)08A | Quantum Mechanical Circuit Variations | FIPS HWCSPRNG | Health Tests (ATECC608A) |

Both the STM32F4 [354] and MKW22D [288] chips supply a TRNG that gathers entropy from sampling multiple free running and jittering oscillators, followed by a post processor based on a linear shift register that ensures statistically well distributed numbers. They also cover basic runtime health tests implemented in hardware, as proposed by NIST [373]. Although the data sheets claim to pass the NIST statistical test suite, the manufacturer NXP recommends against its direct use for cryptographic applications in place of an approved CSPRNG:

> *"It is important to note there is no known cryptographic proof showing this is a secure method of generating random data. In fact, there may be an attack against this random number generator if its output is used directly in a cryptographic application. The attack is based on the linearity of the internal shift registers. Therefore, it is highly recommended that this random data produced by this module be used as an entropy source to provide an input seed to a NIST-approved pseudo-random-number generator based on DES or SHA-1."*
>
> — NXP [288]

The nRF52840 by Nordic [287] implements a TRNG based on sampled thermal noise followed by an optional post processor that reduces bias. Even though the reference manual describes the mechanism as suitable for cryptographic purposes, our results indicate a slightly different picture without the post processor, as we will show later in this section.

The Texas Instruments CC2538 microcontroller [362] implements a PRNG in hardware (HWPRNG), which internally consists of a 16-bit shift register. Hence, its period is limited to $2^{15}$, in contrast to TRNG peripherals of the previous microcontrollers The HWPRNG is seeded by noise samples on the receive path of the on-chip radio. A similar approach is established on the standalone AT86RF233 transceiver module [31], which produces all random values by observing noise from the radio.

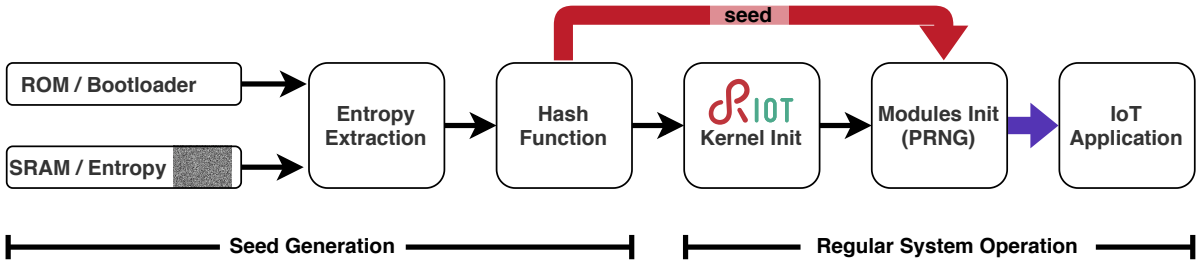



Figure 9.2: PUF SRAM random seeder integration in RIOT.

The ATECC508A [261] is a feature-rich cryptographic co-processor that runs a NIST-approved CSPRNG (HWCSPRNG) combined with an internal seed which is inaccessible from the outside. Thus, we consider it as hybrid device (cf., Section 9.2.4). The seed is automatically updated on every power or sleep cycle. It can also be updated on demand. The seed is generated internally based on entropy extraction from quantum mechanical variations of the circuitry:

> *"In the crypto devices, the random seed comes from variations at a quantum scale within the device. The inherent quantum mechanical entropy of the circuitry within the device provides a truly random seed."* — Atmel [30]

Table 9.2 also lists the SAMD21 and the MEGA2560 microcontrollers by Atmel as examples of the numerous off-the-shelf devices, which completely lack hardware based random sources. This class of devices heavily benefits from external co-processors as well as internally generated entropy from physical resources that can seed an approved software PRNG.

### 9.5.1 SRAM PUF Seeder

Physically unclonable functions (PUFs) are one solution to generate unpredictable numbers even without dedicated electronic circuits. They extract unique output from individual hardware properties. Here, we focus on SRAM PUFs because this memory technology is present on almost all available microcontrollers. Transistor variations of memory cells lead to varying states after device power-on. The startup state of multiple memory blocks form a device-unique pattern plus additional noise, which can be extracted, compressed and used as PRNG seed values [376, 326, 210, 194].

#### 9.5.1.1 OS Integration

**Seed Generation.** The mechanism of a PUF-based seeder is visualized in Figure 9.2 for the example of RIOT. It operates during system startup prior to OS kernel initialization and reads out uninitialized SRAM cells. A PUF measurement is compressed by the lightweight DEK hash to build a high entropy 32 bit number that seeds a PRNG during its instantiation, afterwards in

Table 9.3: Min. Entropy and Hamming Weight between 50 reads of 1 kB SRAM on five SAMD21 MCUs (A–E) at ambient temperature.

| | Device | | | | |
|---|---|---|---|---|---|
| | A | B | C | D | E |
| (i) Entropy [%] | 4.2 | 5.5 | 5.3 | 4.7 | 5.5 |
| (ii) Weight [%] | 50.7 | 49.5 | 51.3 | 49.8 | 53.1 |

Table 9.4: Fractional Hamming Distance from 50 reads of 1 kB SRAM between five SAMD21 MCUs at ambient temperature.

| | Device Pair | | | |
|---|---|---|---|---|
| | A–B | A–C | A–D | A–E |
| (iii) Distance [%] | 49.2 | 49.8 | 50.1 | 50.4 |

the OS startup sequence. It is noteworthy that the lightweight DEK hash is not a cryptographic function. Furthermore, 32 bit entropy is not sufficient for cryptographic use, but the mechanism is extensible to cryptographic requirements.

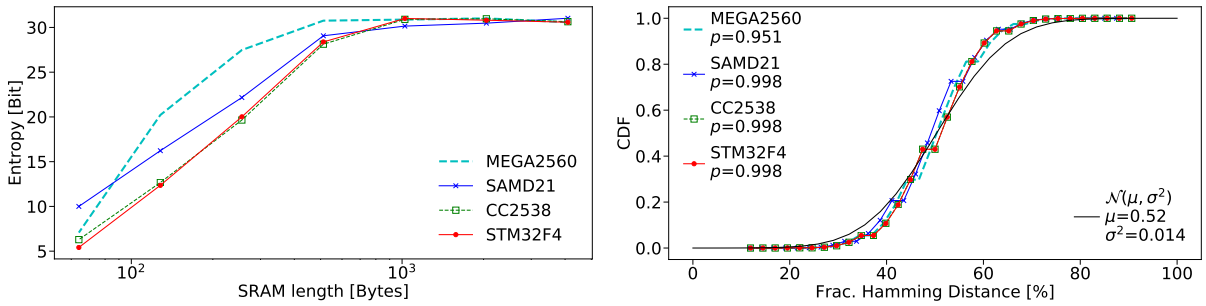
**Re-seed Power Cycle.** Only startup state of uninitialized memory after re-powering provides high entropy values. Either a power-off cycle or a low-power cycle without memory retention is required to generate a new seed. Among others, the required minimum power-off time depends on ambient conditions, age and properties of the power source. In our experiments, an off-time of 1 sec. has proven suitable for all platforms. Schrijen *et al.* [332] analyze the impact of environmental and experimental conditions on the SRAM PUF properties in greater detail.

Nevertheless, soft resets or low-power cycles with memory retention can occur and trigger the startup routine that should not perform a new memory measurement in the absence of a power-off cycle. For a solution, a simple soft-reset detection mechanism writes a randomly chosen marker at a known and reserved address in the SRAM. During the subsequent startup, this memory address is inspected, and a new memory measurement is only performed if the marker has disappeared. Otherwise, a soft-reset counter is incremented, added to the previously generated seed and the result is hashed again for creating a new seed value, which is then stored for the next cycle. The last seed, as well as the soft reset counter could be stored in protected memory for crypto-safe operations.

#### 9.5.1.2 Evaluation of the SRAM PUF Seeder

Next, we analyse SRAM properties on common off-the-shelf microcontrollers and present results for the SAMD21 in Tables 9.3 and 9.4.

**Memory Properties.** As a first step, we inspect the random properties of the memory in detail. We analyse intra- and inter-device variations between multiple PUF responses at ambient conditions. Therefore, we calculate (i) the minimum entropy as a measure of randomness and (ii) the hamming weight to determine bias between multiple startups of one device as well as




(a) Minimum Entropy

(b) Distributions of Hamming Distances

Figure 9.3: PUF SRAM seed evaluations. Min. Entropy for varying input lengths (left) and distributions of fractional Hamming Distances (right).

(iii) the fractional hamming distance between different device responses to quantify inter-device uniqueness. Results for (i) and (ii) indicate existence of a relative min. entropy around 5 % and unbiased patterns. A relative fractional distance of approximately 50 % in (iii) indicates uniqueness of device responses.

**Seed Properties.** For determining the proper size of memory used for seed generation we evaluate the minimum seed entropy for varying input lengths and different platforms as visualized in Figure 9.3 (left). All measurements converge to approximately 31-Bit entropy with 1 kB SRAM.

Next, we test the distribution of hamming distances between multiple generated seeds on every device against normal distribution, using a simple KS-test. The probability function in Figure 9.3 indicates that seed distances on all devices follow a normal distribution with an average expectation value of 0,52 which is slightly biased by the influence of the MEGA2560 platform. Still, all four controllers pass the test with probability values greater than 0,95, which allows to accept that they are unique and uncorrelated.

### 9.5.2 Statistical Analysis with NIST STS

We now evaluate the statistical properties of all hardware based random sources from Table 9.2 by applying the NIST test suite. Following the input size recommendation for each test [43], we test 100 Mbit of random data. Every test is repeated 100 times, which results in 1 Mbit test sequences. Random integers are generated on the constrained device and fed into the evaluation tool over UART. It is worth noting that already the serial transmission of the data takes at least 45 min. with a baud rate of 115200. This added to the randomness generation time which led to experiment times ranging from one to over two hours per run. The experiment was executed under office conditions at ambient temperature.

Results of the $\chi^2$-test on the distribution of $p$-values are shown in Figure 9.4. Certain tests produce multiple $p_2$-values, and we show the average in that case. We display passing tests

with gray bars, and we highlight failed tests with hatched red bars or red arrows in case that the $p_2$-value is too small to be displayed. The test suite additionally calculates a proportion of passed test runs and evaluates its significance. Passing tests in Figure 9.4 have a significant proportion within the confidence interval calculated from $\alpha = 0.01$.


Figure 9.4: Hardware Generated Random Numbers: $\chi^2$-test results on the distribution of probability values from 15 NIST STS tests.

All generators but the nRF52840 (w/o bias correction) and the CC2538 show good statistical properties and pass the test suite. The STM32F4 indicates one failure for the Block Frequency test which analyses the proportion of ones in blocks of 128 bit length. A repeated experiment led to similar results, so we consider this behavior a weakness of that TRNG.

The nRF52840 provides an optional bias correction that is applied to the sampled noise. With enabled correction the nRF52840 (w/ bias correction) passes all statistical tests without deficits. If the post-processor is disabled to increase performance, it fails several of the test statistics as visible in Figure 9.4. As we will see later, the bias correction requires notably more system resources. We repeated the statistical experiment with and without bias correction multiple times with similar results.

The CC2538 consists of a simple 16-bit linear shift register HWPRNG that (i) shortens the generator period (see Section 9.2.3) and (ii) can be attacked with sparse processing resources due to the linearity of the internal shift registers. Correspondingly, it fails in most of the tests.

Table 9.5: Hardware Generated Random Numbers: Throughput and processing time per integer (#).

| Randomness on STM32F4 | Rate [kB/s] | Avg. time per # [$\mu$s] | $\sigma$ time per # [$\mu$s] | Other Platforms | Rate [kB/s] | Avg. time per # [$\mu$s] | $\sigma$ time per # [$\mu$s] |
|---|---|---|---|---|---|---|---|
| STM32F4 | | | | MKW22D | 316 | 33.08 | 0.08 |
| + PUF SRAM | – | 296.46[a] | 0.16 | nRF52840 | | | |
| + ATECC508A | 3 | 11609.69[b] | 889.96 | w/o correct. | 15 | 246.04 | 2.25 |
| + AT86RF233 | 25 | 150.48 | 0.08 | w/ correct. | 6 | 600.28 | 57.03 |
| + TRNG | 1994 | 1.95 | 0.04 | CC2538 | 503 | 6.25 | 0.37 |

[a]Does not include time for power-off cycle [b]ATECC508A always produces 32 Byte blocks

### 9.5.3 Performance Analysis

**Throughput.** We measure the throughput and generation time of different random sources on the STM32F4 platform as well as the internal generators of the other devices listed in Table 9.2. Our test measures the throughput as a single-threaded application that generates streams of pseudo-random numbers continuously, and we count the number of values produced within an interval of 10 seconds. In addition, we measure the time of a single (blocking) function call that returns a random integer with an oscilloscope by toggling an I/O pin via direct register access on the test device. In this way the measurement overhead remains negligible. The ATECC508A crypto-chip always processes 32 Bytes per request even if only integer values are requested. We display rates, average processing times per random integer, and their standard deviation in Table 9.5.

Clearly, the PUF based seeder as well as the externally connected devices perform two up to four orders of magnitudes slower in comparison to the on-chip generator of the STM32F4 that returns random values via direct register access. It is important to note that our measurements for the SRAM based seeder only include processing times of the generator itself that consists of the reset detection, memory readout, and hashing to create a final random value. In practice, the required power-off cycle as well as the OS startup are added, which can take tens to hundreds of milliseconds and is heavily dependent on the OS configuration. However, in the all-day use of an IoT-device, low power cycles occur repeatedly. The cryptographic co-processor ATECC508A is notably the slowest, but it is the only candidate that runs a hardware based cryptographically secure random number generator. Its performance strongly relates to the mode of its device driver [262]. In our measurements, we use the polling mode, which queries the device for the completed execution 4–7 times before random data is ready. Alternatively, in the non-polling mode, the driver waits a maximum execution time for each command, after which the data

is ready to be fetched from the co-processor. This potentially increases execution times but leaves room for parallel processes to be scheduled or low-power sleeping. Furthermore, the ATECC508A as well as the AT86RF233 transceiver require additional I2C/SPI transmissions in comparison to the on-chip solutions. Both the driver overhead and the transport of random data are included in our measurements. It is noteworthy, though, that a real-world deployment of the ATECC508A would process random data internally, which relieves transport over the I/O interface and reduces computation demands of software implementations in a crypto stack.

Among different MCUs on-chip generators, the STM32F4 performs fastest. Although its CPU runs with the highest frequency (96 MHz), the processing time is not directly proportional to the CPU speed. In comparison, the MKW22D implements a similar TRNG and runs at half the CPU speed (48 MHz), but takes more than ten times longer to produce a random integer value. The CC2538 unsurprisingly operates comparably fast as it only operates a simple shift register without sampling noise. Both configurations of the nRF52840 generator indicate low throughput of 6–15 kB/s in comparison to the other generators, which presumably relates to the internal sampling procedure. The bias correction leads to statistically good properties of the output sequences, but it reduces computational speed by a factor of 2.5.

**Energy Consumption.** To evaluate the energy consumption of each generator, we measure the current consumption of all hardware based approaches with a digital sampling multimeter (Keithley DMM7510 7 1/2) at 1 MS/s, and we drive the board from an external regulated voltage supply. All development boards provide a measurement header to probe the current that flows to the microcontroller. For the externally connected devices (*i.e.,* ATECC508A and AT86RF233), we additionally measure the current to the power supply pin. Our energy measurements include driver overhead as well as the transmission over I2C/SPI. Measurements for the SRAM seeder include reset detection, memory readout, and hashing, referring to our throughput evaluation. Measurements for on-chip generators include associated testing overheads that are always executed, if available on a hardware platform. Some hardware based random generators execute at the same scale as the sampling resolution, thus, we measure generation of 1000 integers per run in that case and normalize the cumulated values afterwards. We repeat every experiment 1000 times. In setups that involve external hardware, we measure the MCU and the external device separately. If applicable, we separate microcontroller and external device consumption in our graphs.

Figure 9.5 displays our test results on the energy consumption. The radio based approach as well as the SRAM based seeder consume one order of magnitude more energy than the on-chip generated numbers, though, both mechanisms miss an online health testing. The ATECC508A clearly has the highest energy footprint, which is strongly related to the device driver overhead. As depicted in Table 9.5 the co-processor requires more than 11 ms to process the next output, during which the MCU is polling the co-processors status. While co-processing, though, the MCU is free to process other data or to go to a deeper sleep mode for energy saving. As depicted in Figure 9.5, the microcontroller consumed the larger portion of energy up to 190 $\mu$J.

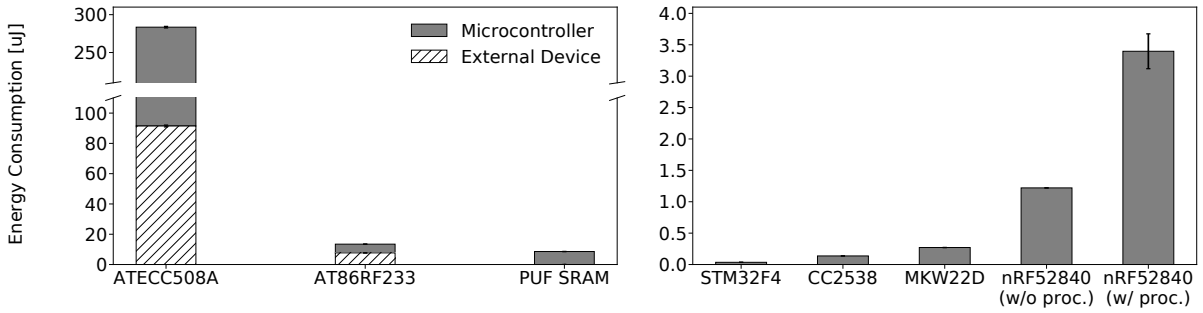



Figure 9.5: Hardware Generated Random Numbers: Energy consumption of external (left) and internal on-chip generators (right).

Furthermore, the ATECC508A produces a minimum of 32 random bytes per request, thus, up to eight integers can be fetched for the same cost. It is worth noting that continuous health tests are also applied to random samples on the external device.

The radio based random generation is second most energy hungry after the cryptographic co-processor. Although this approach is ten times faster, it only saves a factor of four in energy, which is due to the higher current draw of approximately 12.5 mA in receive mode of the AT86RF233 during random bit generation. The ATECC508A on the other hand only drives around 3 mA during operation. Results for the SRAM based seed generator scale similar to the radio based approach. It is worth noting that we only display the PUF SRAM overhead here that comes on top of a power-off cycle. Results for on-chip generators reflect similar properties as the processing times. The consumed energy to produce one random integer is below 3.5 uJ on all internal on-chip generators. All devices except the CC2538 platform apply continuous health tests in the hardware. STM32F4 is notably the most frugal competitor and reduces the consumption per integer to less than 0.35 uJ.

The difference in energy consumption of the two nRF52840 modes attains about a factor of 2.5, which increases only slightly less than the processing times listed in Table 9.5. Still, bias correction introduces a notable increase in processing time and thereby in energy consumption.

## 9.6 Software Generated Pseudo-random Numbers

Since von Neumann's early work [384], the generic method of obtaining pseudo-random numbers builds up on some (deterministic) function that is iteratively applied to a (random) seed and attempts to approximate the output of uniformly distributed, independent random trials. As of today, very many of such pseudo-random number generators exist, which comply to various random quality and complexity requirements. From the perspective of an IoT operating system, we are interested in two types of highly efficient, memory-strained algorithms: (i) an ultra-lightweight general purpose PRNG, and (ii) a crypto-secure PRNG that meets the resource constraints of class 1 IoT devices. In the following, we will introduce and analyse eight popular

generators—four complex generators of high quality and four lightweight candidates for general purpose random generators in the IoT.

We apply the NIST, DIEHARDER, and the TestU01 suites to these PRNGs using RIOT as OS platform. Our test programs run as RIOT native processes on Linux servers to speed up experiments. Thereby, we seeded all generators identically, except for the CTR PRNG and the NIST Hash DRBG, which impose specific requirements on their seeds.

### 9.6.1 Complex Generators

**Fortuna.** The Fortuna PRNG is considered a cryptographic random number generator (DRG.4 [201]) and was designed to overcome the demand for entropy estimators that tend to be complex and inaccurate [109]. Internally, the Fortuna algorithm maintains pools of entropy from which a periodic re-seeding is performed. Pools are filled from different available entropy sources, whose values are distributed among these pools. The entropy accumulation is conducted by hashing the internal generator state and one entropy pool at a time using a SHA-256 function. That mechanism allows generating random sequences of unlimited period, though, it cannot overcome the requirements for proper entropy sources for seeding the generator and for updating entropy pools. Potential side effects of re-seeding have been discussed in Section 9.2.3. The final blocks of pseudo-random output are generated by an AES-128 block cipher in counter mode.

**CTR PRNG.** The CTR PRNG is a cryptographic generator (DRG.4 [201]) specified and approved by NIST [40]. It is based on a 128 bit AES block cipher in counter mode, why it provides 128 bit security strength. For this, the generator requires seeds of at least 128 bits entropy. The TinyCrypt library implementation, which we use in our subsequent evaluation, requires seeds with a minimum length of 256 bits. This is in contrast to all other implementations presented in this section.

**SHA256PRNG.** The SHA256PRNG is a generator that provides cryptographically strong random numbers (DRG.2 [201]). The original mechanism of this generator was introduced in FIPS 186–1 [279] and analyzed by Kelsey *et al.* [187] and Desaj *et al.* [88]. Outputs are generated by hashing the internal generator state, which is updated thereafter by a linear transformation of the hash.

SHA256PRNG is the successor of the SHA1PRNG, which became popular as the choice of the Java *SecureRandom* class. Until recently, it was considered secure [118] but got deprecated in Android N [17]. The reasons for deprecation mainly relate to a seeding bug in Java and to NIST deprecating the underlying SHA-1 hash function because Wang *et al.* [392, 353] discovered an attack that decreased the number of brute-force tries needed to foster state collisions from $2^{80}$ to $2^{63}$ operations. For that reason, SHA256PRNG replaces the SHA-1 output function by SHA-256.

The generator is forward secure up to the previous-to-last value, which can be recovered

from the current generator output by inverting the state update function. SHA256PRNG is confined to a single hash computation per block, which makes it computationally efficient. To add full forward secrecy NIST developed a collection of improved and standardized random bit generators [40], the deployment of which has been recommended in revisions of the FIPS 186 [283, 284] standard.

We briefly discuss results about the NIST Hash DRBG (using the SHA-256 hash) at the side. It extends the SHA256PRNG by a cryptographic function that hashes the internal generator state to harden backtracking resistance after state compromise. This requires two hash computations on each block. The NIST Hash DRBG also implements an approved re-seeding mechanism to achieve unpredictability even after state compromise (DRG.4 [201]).

**Mersenne Twister.** The Mersenne Twister is a widely used generator, which in its default version is known to be non-secure [259], even though crypto-secure variants have been explored successively. Known advantages of that algorithm are a long period and comparably fast operations, because the generation of pseudo-random numbers avoids multiplication and division. Instead, it requires a large internal buffer of 624 integers.

### 9.6.2 Lightweight Generators

**Tiny Mersenne Twister.** The Tiny Mersenne Twister is a derivate of the original Mersenne Twister that adapts to resource constraints on the price of a narrowed scope. It heavily reduces buffer requirements, on the price of a shortened period length. The generator is rather a reduced, memory efficient fallback solution of the full Mersenne Twister [322].

**Xorshift.** The Xorshift generator belongs to the family of linear-feedback shift register generators that are not cryptographically secure. It is known for its resource efficiency as it is confined to simple XOR and shift operations. In its simplest 32 bit state generator, we refer to it in the following. Marsaglia [254] proposed a collection of extended Xorshift PRNGs with an increased period length and improved statistical properties. Vigna *et al.* analyzed and improved these generators further [378, 379, 50]. We briefly discuss results about the Xorshift64* and the Xoroshiro128+ derivatives at the side, which we integrated into RIOT for comparison. The Xorshift64* consists of a 64 bit state and applies a constant multiplication to the output for bit scrambling. The Xoroshiro128+ requires 128 bit state, although it only outputs 64 bit values per cycle. In contrast to Xorshift64* it adds two consecutive state values as a nonlinear transformation to the output.

**Park-Miller "Minimal Standard".** The Minimal Standard algorithm is a linear congruential generators (LCG) and has been known for decades [298]. Motivated by its objective to design a lightweight generator that is confined to 32 bit arithmetic without divisions, it was criticized for its statistical properties repeatedly [395]. The generator has a period or $2^{31} - 1$, and it is limited to produce 31 pseudo-random bits during each cycle. In RIOT, however, the API presumes 32 bit random integers, thus, one integer is generated by combining two generator outputs on each

random request, which limits the usable period to $2^{30} - 1$. LCGs are generally not designed for cryptographic purposes.

**Knuth LCG.** The Knuth LCG is a widely used linear congruential generator, which is computationally lightweight and has been examined for decades. It is implemented in a range of software projects and the source code is available in different standard libraries such as Newlib [383] or Musl C [277]. The generator adopts a multiplier that was obtained by Knuth [205, Chapter 3.3.2, p. 108] but current implementations differ from the "MMIX" Knuth LCG in its increment. Furthermore, it truncates the most significant bits of the 64 bit state to 32 bit output values due to known poor statistical properties of the lower bits in modulo-2 generators [252].

### 9.6.3 Statistical Analysis with NIST STS

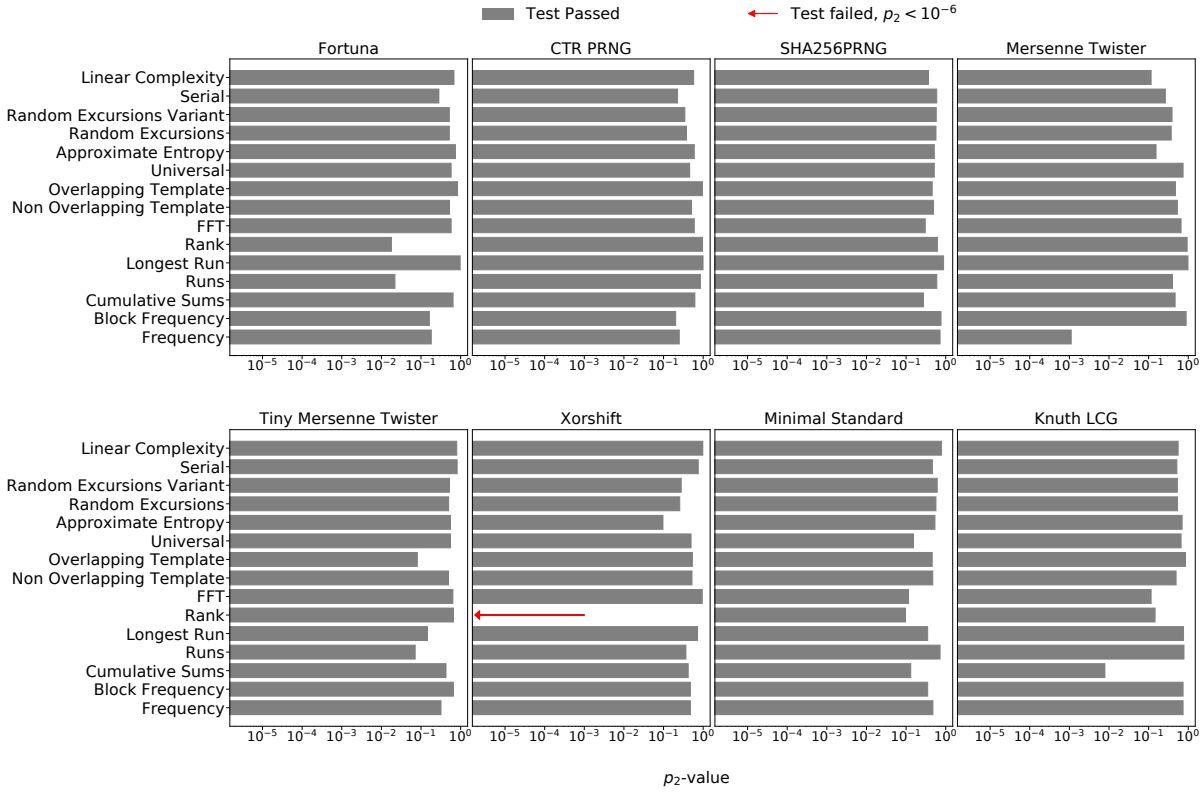



Figure 9.6: Pseudo Random Numbers: $\chi^2$-test results on the distribution of probability values from 15 NIST STS tests. $p_2$-values$\geq 10^{-4}$ pass the hypothesis of uniformity.

Our first statistical analysis applies the NIST STS test suite to these generators. The results of the $\chi^2$-test are shown in Figure 9.6. Analog to Section 9.5.2, tests are only passed after a significant proportion of repeated successful runs. That is, 96 successful results for a sample size of 100 sequences. Test 11 and Test 12 (Random Excursions and Random Excursions Variant), however, are not always applicable and as such they reduce the sample size internally. In our

configuration, this led to proportions of sequences passing one of both tests in ranges from 53/56 to 67/71 in different PRNG measurements.

All generators except for the Xorshift pass all 15 statistical NIST tests. Xorshift flaws in the Rank (Binary Matrix Rank) test. This test analyses linear dependencies among substrings of a sequence. In our case, all 100 test sequences fail already the first order hypothesis test and thus, the $\chi^2$-distributions test fails in consequence. We consider this an algorithmic weakness. The deficit is fixed by the extensions applied to Xorshift64* and Xoroshiro128+, which pass all tests.

The Frequency Test on the Mersenne Twister had a very small $p$-value on the first run (not displayed here) and we ran the same test again with altered seed value during initialization. This succeeded the test as displayed in Figure 9.6. We want to stress that the magnitude of a $p$-value in hypothesis testing is not a causal measure of quality. This means that a low $p$-value confirms to reject a null hypothesis (sequence is not random) but it does not claim how likely it is that the alternative hypothesis is true (sequence is random). For further interpretation of statistical tests, we refer the reader to the NIST test suite description [43] and work by Greenland *et al.* [134].

### 9.6.4 Statistical Analysis with DIEHARDER

Next, we apply the DIEHARDER test suite to our random generator candidates. DIEHARDER test values behave similar to the NIST STS results presented in the previous section, but procedures are more demanding and failures are more distinctive.

Results of the Kolmogorov-Smirnov test for each PRNG in RIOT are displayed in Figure 9.7. Similar to the NIST tests, we plot average $p_2$-values, where applicable. In the style of previous graphs, we display passing tests with gray bars, and we highlight failed tests with hatched red bars or red arrows in case that the $p_2$-value is too small to be displayed. Additionally, we mark weak results with black bars. All complex generators displayed in the first row of Figure 9.7 (*i.e.,* Fortuna, CTR PRNG, SHA256PRNG, and Mersenne Twister) pass all tests. Several failures must be observed for the lightweight generators displayed in the second row of Figure 9.7. One test returns weak results, which is expected in 1 % of the test cases due to the uniform distribution of p-values. The DIEHARDER help page [60] recommends repeated test executions and analysis of p-value histograms on weak results. All weak results passed a repeated experiment run with a different seed.

Several failures must be observed for the lightweight generators displayed in the second row of Figure 9.7 (*i.e.,* Tiny Mersenne Twister, Xorshift, Minimal Standard, Knuth LCG). The Tiny Mersenne Twister fails Tests OQSO (Overlapping Quadruples Sparse Occupancy Test), and the DNA test, which both examine the distribution of overlapping substrings in a stream of random integer values. These results indicate a systematic problem of the generator. The Xorshift generator fails the Monobit 2 test, the 32x32 Binary Rank test, and the Count the 1s Stream

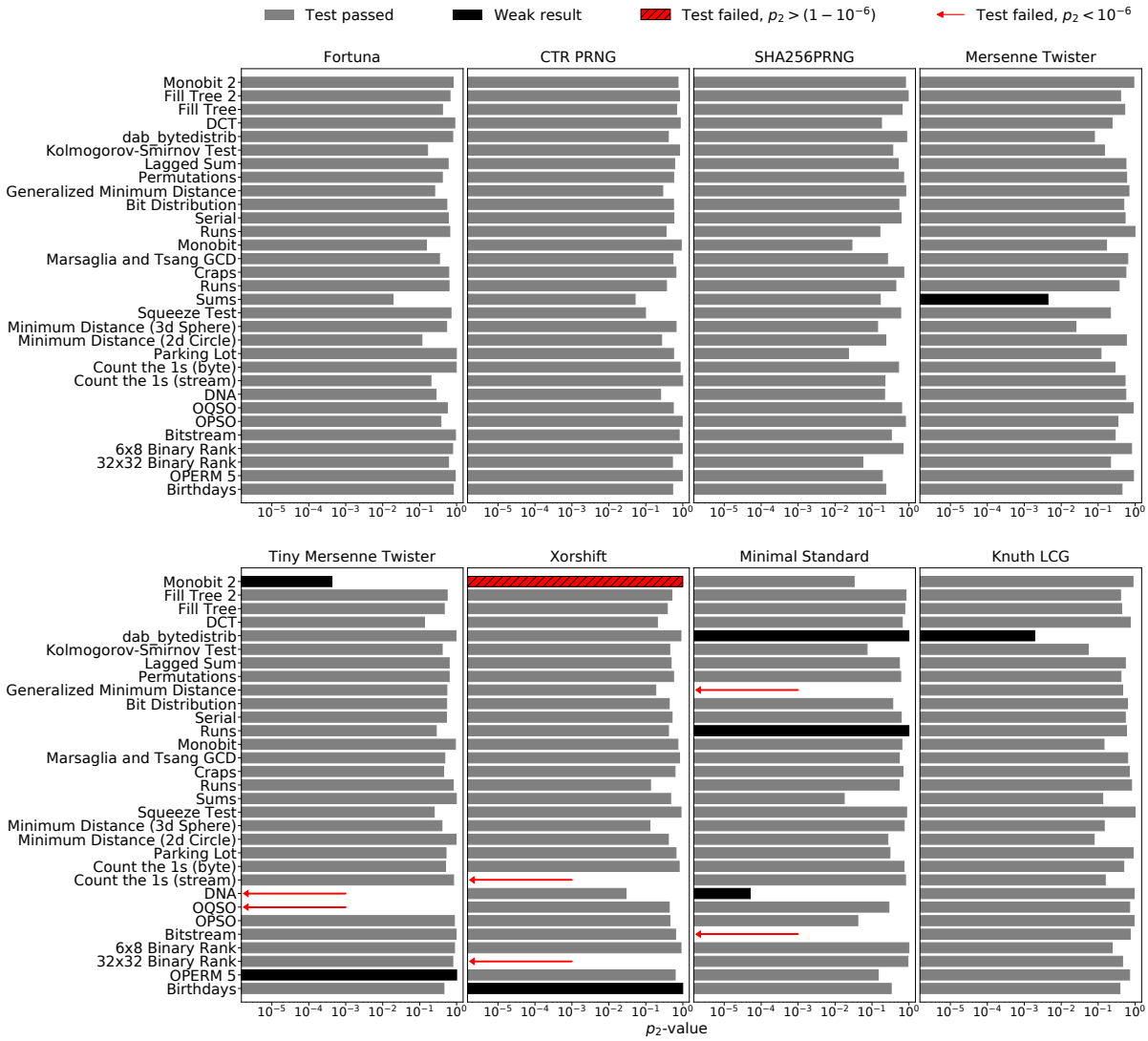



Figure 9.7: KS-test results on the distribution of probability values from 31 DIEHARDER tests. $p_2$-values in the significance interval $(\alpha, 1-\alpha)$ with $\alpha = 10^{-6}$ pass the test. $p_2$-values within the interval $(\alpha_2, 1-\alpha_2)$ with $\alpha_2 = 5 \cdot 10^{-3}$ are considered as weak.

test. The Monobit 2 is a derivate of the NIST Frequency test and measures the proportion of zeros and ones within blocks (12-bit blocks applied here). Surprisingly, the general Monobit test, which counts "1" bits in a long sequence of random samples (100000 samples considered here) does not fail. Hence, the Xorshift introduces bias within smaller sub-blocks, which is compensated over the whole sequence. Results persist after repeated experiment executions with different seeds. Failing on the matrix rank test was expected, as the equivalent from the NIST STS already failed in a similar configuration. The Count the 1s Stream Test examines whether the distribution of ones in a stream of bytes matches that of uniform random bytes (Binomial(8, 0.5)). Failures indicate that Xorshift output streams produce repeated "words"

Table 9.6: Summary of test results from the "BigCrush" of the TestU01 environment.

| Generator | Fortuna | CTR PRNG | SHA256 PRNG | Mersenne Twister | Tiny Mers. Twist. | Xorshift | Minimal Standard | Knuth LCG |
|---|---|---|---|---|---|---|---|---|
| Failures | 1/160 | 0/160 | 0/160 | 2/160 | 13/160 | 58/160 | 71/160 | 9/160 |

that appear with pronounced probability. It is likely that the same effect led to bad results of the linear dependency test among sub-matrices. The advanced Xorshift64* and Xoroshiro128+ generators pass all DIEHARDER tests.

The Minimal Standard generator fails the Bitstream and the Generalized Minimum Distance test. The first successively analyses overlapping 20-bit tuples ($2^{20}$ possible words) and tests the statistic of missing words for a normal distribution. Failing this test indicates recurrence of patterns with enhanced probability. The second test places random pairs of points in a square and tests its squared distances for an exponential distribution. The Minimal Standard generator fails to produce outputs that appear independent in this dimension. Finally, this generator issues a suspiciously high number of weak test results. For further interpretation of test results, we refer to the DIEHARDER Test Suite description [61].

### 9.6.5 Statistical Analysis with TestU01

We additionally apply the "BigCrush" from the TestU01 test suite to all software generators. Table 9.6 summarizes test results with failures reflecting the number of reported test statistics with $p$-values outside the confidence interval [0.001, 0.9990].

Results reflect a similar picture as depicted by NIST and DIEHARDER. TestU01, though, stresses more failures due to a higher number of tests and tighter hypothesis criteria. According to L'Ecuyer *et al.* [225], $p$-values outside the significance interval are obtained approximately 2 % of the time, even if the PRNG behaves well. This, however, should not reoccur systematically.

The Minimal Standard misses almost half of all tests (45 %), and the Xorshift fails a surprisingly high number of 58 tests (35 %), which reduces to 1–2 failures for its enhanced derivatives Xorshift64* and Xoroshiro128+. The Tiny Mersenne Twisters attains a failure rate of 8 %, which still exceeds the acceptable rate by a factor of four, whereas the Knuth LCG performs notably better, failing about 5 % of the test statistics.

For the Mersenne Twister, both failures have a $p$-value of more than $(1-10^{-15})$, which significantly misses the acceptance interval, and unacceptable results reappear in repeated experiments with different start values. This indicates systematic weaknesses. In contrast, all CSPRNGs report zero or singular failures based on $p$-values at the order of $10^{-4}$, which disappear for repeated tests.

### 9.6.6 Performance Analysis

In the constrained IoT, an important dimension for any base system primitive lies in its performance. To assess the value of the different random number generators, we measure the computational speed, the memory overhead, and the energy consumption of all pseudo-random number generators. Thereby, we consider base mechanisms, and we disable re-seeding, if available.

**Throughput.** We measure the throughput and speed of each generator on the STM32F4 hardware platform that has been introduced in Section 9.5. Our test applications are implemented like presented in Section 9.5.3. In addition, we display maximum values for cases, in which a generator occasionally takes significant time to rebuild its internal state. Results are summarized in Table 9.7.

Table 9.7: PRNG throughput and processing time per integer (#) measured on STM32F4.

| Generator | Rate [kB/s] | Avg. time per # [$\mu$s] | Max. time per # [$\mu$s] |
|---|---|---|---|
| Fortuna | 44 | 87.50 | – |
| CTR PRNG | 102 | 442.01[a] | – |
| SHA256PRNG | 393 | 10.04 | 69.60 |
| Mersenne Twister | 3605 | 0.85 | 189.20 |
| Tiny Mers. Twist. | 4807 | 0.62 | – |
| Xorshift | 8152 | 0.28 | – |
| Minimal Standard | 3348 | 0.98 | – |
| Knuth LCG | 6147 | 0.44 | – |

[a]CTR PRNG calculates at least one AES-128 cipher per call

Naturally, the four lightweight generators are fastest. They can reliably compute a random number in less than a microsecond. The two ultra-lightweight algorithms Knuth LCG and Xorshift can even produce several numbers per microsecond on the constrained microcontroller, which must be considered a very low runtime overhead. With a production rate of more than five MB/s, these generators could seamlessly support a stream cipher, if they were cryptographically secure. Unfortunately they are not, but showed some weaknesses in statistical tests as discussed in the previous section. Xorshift clearly has the highest throughput. Its derivatives Xorshift64+ and Xoroshiro128+ perform around 2–3 times slower.

From the complex generators, SHA256PRNG and the Mersenne Twister are presented with additional maximal values in processing time. For SHA256PRNG this is due to the algorithmic property of producing 32 Bytes in one hash call, which thereafter are split into 8 Byte long integer values. Hence, a hash value is computed only every fifth call, which then takes signif-

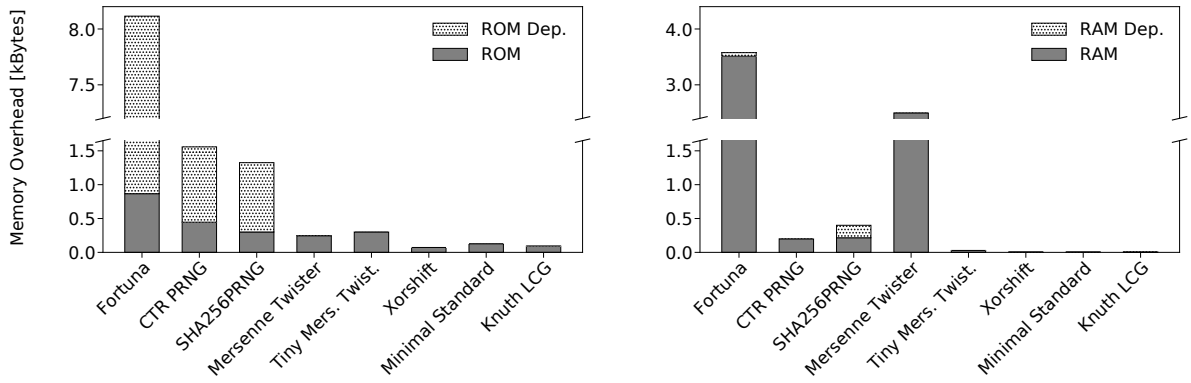
icantly longer. The Mersenne Twister follows a similar approach, although it does not involve cryptographic functions. At the initialization and every 624 calls, it generates 624 fresh pseudo-random integers, which takes up to 190 $\mu$s. This is orders of magnitudes longer than simply returning a value from its buffer. This notably degrades its performance in comparison with the other lightweight generators.

The three cryptographically secure generators Fortuna, CTR PRNG and SHA256PRNG operate more than one order of magnitude slower than the general purpose counterparts. All algorithms involve cryptographic functions (AES-128, SHA-256), which are computationally expensive on constrained microcontrollers. Clearly, the SHA256PRNG is the most vital among all crypto-generators presented, requiring around 70 $\mu$s on every hash computation, which leads to an average time of 10 $\mu$s per integer with number caching, and a rate of almost 400 kB/s. The more advanced NIST Hash DRBG (SHA-256) doubles the time per integer due to additional hash computation in the feedback path. The implementation, however, does not allow number caching. This overhead is compensated while requesting larger random blocks or streams because state updates only occur once after each API call, while generating large outputs requires multiple hashing.

In comparison to SHA256PRNG, the CTR PRNG generates random four times slower in a stream of numbers, whereas returning one random integer takes between six and thirty times longer. The CTR PRNG on the other hand has an almost constant runtime per integer. It is noteworthy that up to 16 Bytes can be obtained by the CTR PRNG without runtime overhead because this generator computes one AES-128 block on every call that delivers 128 Bit from which four 32 Bit integers can be created. The Fortuna operates slowest when requesting continuous random data, and it halves the throughput of the CTR PRNG. It has a constant runtime per integer of less than 90 $\mu$s, which on the other hand is faster by a factor of five than the CTR PRNG. It requires approximately 120 % of time as the SHA256PRNG when processing a new hash internally (every eighth call), why the rate is less by a factor of almost ten.

The different design choices between SHA256PRNG and CTR PRNG relate to security implications. Holding a precomputed hash value in RAM as SHA256PRNG does while only one integer is requested may violate security requirements. In the presence of protected memory, a generator could secure its state and its cached numbers to reduce the attack surface. Low-cost devices often lack memory protection mechanisms and as such, a design might be favored that avoids keeping sensitive data in memory for longer intervals in order to prevent (i) manipulation of future output (see Section 9.1) as well as (ii) predicting future random numbers (see Section 9.2). In the presence of frequent re-seeding from fresh entropy, the generator state becomes less sensitive to memory attacks. In contrast, the purpose of caching is performance enhancement and random numbers would be simply returned without update after re-seeding. Still, the performance per integer request of the CTR PRNG would notably benefit from a caching mechanism.

**Memory Overhead.** Memory is a particular scarce resource on IoT devices, why memory of




(a) ROM (b) RAM

Figure 9.8: PRNG memory overhead in ROM (left) and RAM (right) measured on the STM32F4 microcontroller. "Dep." denotes memory requirements of dependent software modules (*i.e.,* hashes, ciphers) that a generator may include.

random number generators should have the lowest possible footprint. We evaluate the memory overhead, which comes on top of a minimal RIOT build while enabling different PRNGs at compile time for the target STM32F4 MCU in Figure 9.8. Numbers are differentiated w.r.t. RAM and ROM memory. Furthermore, crypto-purpose generators include dependencies such as hash functions and ciphers, which are highlighted as "RAM/ROM Dep.".

Results reassure the unsurprising previous observation that complexity of the PRNGs correlates with resource consumption. Similar to the throughput measurements, Xorshift, Minimal Standard and Knuth LCG generators remain most frugal in memory and only require around 100 Bytes additional ROM and few additional Bytes in RAM. The internal state size (32 bit), *i.e.,* 4 Bytes in RAM suffice for the Xorshift, whereas the Knuth LCG adds an internal multiplier to a total of 8 Bytes RAM. Similarly, Xorshift64* has an 8 Byte state, but it requires additional memory to split and buffer 8 Bytes generated on each cycle into two 32 bit integers. In total, Xorshift64* allocates 20 Bytes RAM and Xoroshiro128+ allocates 28 Bytes. The Xorshift-derivatives hence operate on a different scale than the ultra-lightweight alternatives. Both Mersenne Twisters allocate between 250–300 Bytes in ROM, whereby the "tiny" version surprisingly requires the higher amount. The Mersenne Twister is comparably RAM intensive by allocating up to 2.5 kB, which is mainly used for its buffer of 624 integers.

Crypto-secure PRNGs such as SHA256PRNG and CTR PRNG are rather efficient in RAM when compared to the Mersenne Twister, but are more demanding in ROM memory. This is mainly due to its dependent cryptographic functions. The NIST Hash DRBG (SHA-256) uses RAM similar to the SHA256PRNG, but it requires additional 800 Bytes ROM. The reasons relate to its extra logic in the state update function as well as its re-seeding capabilities. The most demanding generator in terms of ROM memory is the Fortuna due to its dependencies

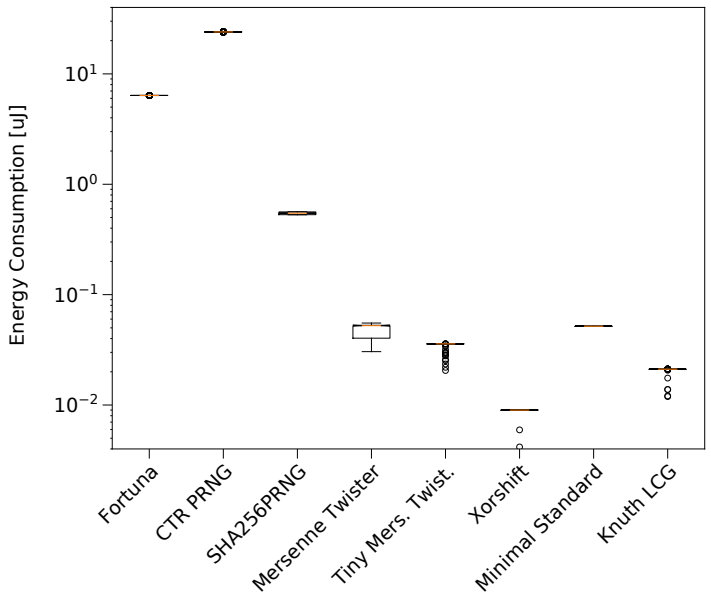



Figure 9.9: PRNG energy consumption per integer measured on a STM32F4 microcontroller.

to multiple cryptographic functions. It includes the SHA-256 and the AES-128 RIOT modules, which is the main source of high ROM requirement. Furthermore, the Fortuna internally implements an entropy pool that by default initializes 32 SHA-256 contexts, one of which requires 104 Bytes. This is the main reason for its high RAM consumption of around 3,5 kB. Here it must be stressed that this already consumes 10 % of the available memory on the test device, or 44 % on an Arduino Mega 2560.

**Energy Consumption.** We evaluate the energy consumption with the same setup as described in Section 9.5. Figure 9.9 displays the energy consumed by each pseudo-random generators in RIOT, measured on the STM32F4 platform. The Fortuna, CTR PRNG and the SHA256PRNG crypto-purpose generators are most expensive in energy—up to several micro Joule per integer. Thereby, SHA256PRNG operates at the lower end and the SHA256-based NIST Hash DRBG consumes about twice as much. The CTR PRNG notably demands maximal resources. Using approximately 24 $\mu$J it drains a factor of 3.7 more energy than the Fortuna. The CTR PRNG generates four integers in one call, why it outperforms the Fortuna slightly in a stream of random numbers.

All other PRNGs operate on the same scale as the on-chip TRNG on that device (compare Figure 9.5), which takes approximately 0.03 $\mu$J. Non-cryptographic generators remain by a factor of 20–100 below the most efficient cryptographic generator SHA256PRNG. The most resource-friendly general-purpose PRNGs are the Xorshift with an average of 0.01 $\mu$J and the Knuth LCG with 0.02 $\mu$J per integer, which is in agreement with our findings from throughput measurements in Table 9.7. The enhanced Xorshift64* and Xoroshiro128+ consume 3–4 times more energy

Table 9.8: Summary of the security properties vers. performance trade-offs for CSPRNGs. Backward secrecy can be enabled with external entropy source (✚). Resource consumption is high ( ↑), medium (→), or low (↓).

| | Security Properties | | | Performance | | | |
|---|---|---|---|---|---|---|---|
| Generator | Statistic | Forward | Backward | Runtime | ROM | RAM | Energy |
| Fortuna | ✓ | ✓ | ✚ | → | ↑ | ↑ | → |
| CTR PRNG | ✓ | ✓ | ✚ | ↑ | → | → | ↑ |
| SHA256PRNG | ✓ | ✓[a] | ✚ | ↓ | → | → | ↓ |

[a] Previous-to-last random value not protected by forward secrecy.

than their lightweight 32 bit Xorshift companion, which is a notable increase over the Knuth LCG.

### 9.6.7 Recommendations on PRNGs

Having examined eleven widely available software-PRNGs (eight algorithms plus three variants) that cover the basic levels and functions, we are now ready to make the choice of recommendation for random number generators to be included in a constrained IoT operating system. Following the previous discussion in Section 9.3, we differentiate our selections in one general purpose and one crypto-secure PRNG as two different system functions.

**General Purpose Generator.** The general purpose generator should be very lightweight, while complying with common statistical requirements. It should run seamlessly on very constrained 8 bit microcontrollers at low energy costs.

From the two ultra-lightweight generators, only the Knuth LCG passes all NIST and DIEHARDER statistical tests. It performs second best. The Xorshift generator consumes about half of its resources, but has notable statistical flaws. The Knuth LCG produces better output sequences while being similarly fast. The enhanced Xorshift64* or Xoroshiro128+ generators exceed resource consumption of the Knuth LCG significantly. All other lightweight generators admit lower statistical quality at higher cost. We therefore recommend the Knuth LCG to be used as general-purpose, non-cryptographic PRNG in the constrained IoT.

**Crypto-secure Generator.** The Fortuna, SHA256PRNG, (NIST Hash DRBG, ) and CTR PRNG are the only candidates for a CSPRNG, as they are constructed from secure and non-invertible cryptographic functions. Table 9.8 summarizes the security properties and performance and illustrates the design trade-offs as a basis of our recommendation.

The CRPRNGs pass all statistical tests (including BigCrush) and fulfill the requirements of unpredictability and brute-force resistance. Perfect forward and backward secrecy after a state compromise is assured by the Fortuna, CTR PRNG and the NIST Hash DRBG based on their one-way nature during state update in combination with re-seeding. The SHA256PRNG holds

a (linear combination of) the previous-to-last state and thus only guarantees forward secrecy for earlier values due to its secure one-way SHA-256 output function (cf., [187]). Backward secrecy can be added by re-seeding on demand to all generators as discussed in 9.2.3 using predefined interfaces to entropy sources.

The SHA-256 generator stands out as it is the only cryptographically secure algorithm that attains moderate performance values. All competitors exceed the SHA-256 performance measures by one order of magnitude in at least one dimension. We conclude that the resource frugality in combination with modular cryptographic robustness justify to commend SHA256PRNG as the CSPRNG in the constrained IoT.

## 9.7 Random Numbers on AI Platforms

Machine learning and other algorithms of Artificial Intelligence (AI) recently gained attention and are considered for deployment at the IoT Edge. They shall serve use cases such as voice recognition, object counting, or anomaly detection. Machine learning, in particular reinforcement learning makes heavy use of random numbers, since randomized algorithms (*e.g.,* Monte-Carlo methods [306]) are involved to explore state spaces. Software libraries for machine learning evolve and exist already, optimized for constrained embedded devices [84, 35].

High processing demands triggered a new set of hardware platforms to facilitate computation of AI algorithms at minimal energy consumption. A common approach is represented by dual-core SoCs and hardware accelerators to offload the main CPU from processing. We identified the following three categories in the data sheets: signal processors, neural network accelerators, and tailored hardware engines for assisted audio-video processing. Commercial products largely base on ARM Cortex-M processors with a proprietary instruction set architecture (ISA). The academic community introduced a series of open source RISC-V processors that vary with dedicated purposes while targeting low energy consumption [328]. Parallel ultra low power (PULP) processors feature an optimized processor design. Multiple cores can be clustered and share a coarse-grained memory architecture, the instruction cache and peripherals. Energy efficiency is achieved by speedup after parallelization and operating the cores 'near threshold' by applying voltage and frequency gating [122]. This extends utility in the contexts of signal-, neural network-, or audio-video processors. PULP clusters can be augmented [77] by cryptographic hardware engines to facilitate low-overhead encryption of network traffic, or by convolution engines.

The modular design and performance of these novel RISC-V based processors have been analyzed and simulated by De Giovanni *et al.* [125]. They observe a speedup factor of five and energy savings of 40 % for an 8-core PULP cluster over a single core. Additional hardware acceleration decreases energy consumption down to 50 %. Their findings are in agreement with Wang *et al.* [393], who analyzed neural network inference on constrained IoT devices. They compare the processing overhead and energy consumption of a machine learning algorithm

Table 9.9: Overview of the different processor cores on the RV32M1 microcontroller. It runs at 48 MHz and provides 384 kB RAM, 1.2 MB internal and 4.0 MB external flash memory. The naming scheme IEFCM resolves to the following architectural components. I: base integer instruction set, E: embedded base integer instruction set, F: single precision floating point extension, C: extensions for compressed instructions, M: integer multiplication and division extension.

| Processor Core | Instr. Set Architecture | Pipelining [# stages] | Special Features |
|---|---|---|---|
| ZERO-RISCY | RV32-IECM[a] | 2 | Area optimized (2.2 x smaller than RI5CY)<br>Energy boost for mixed control/arithmetic code |
| RI5CY | RV32-I(F)CM[b] | 4 | Post-incrementing load and stores<br>Single-cycle multiply-add & ALU extensions<br>Auto increment hardware loops<br>Memory protection unit |
| ARM Cortex-M4F | ARMv7E-M[c] | 3 | Branch speculation engine<br>Single-cycle multiply-add extension<br>Memory protection unit |

running on an off-the-shelf ARM Cortex-M4 node and two optimized RISC-V platforms, one single-core (RI5CY) and one 8-core cluster of RI5CY processors (*Mr. Wolf* IoT processor [309] for high processing demands). Their results indicate that RI5CY outperforms Cortex-M by a factor of up to 1.3 for fixed point integer arithmetic. The neural network use case provides a speedup of six times and reduces the energy consumption by up to 70 % on the multi-core platform in comparison with a single-core Cortex-M4 operation.

The shift of computational complexity towards edge devices imposes new requirements on the random number generation. The rate of statistical (pseudo-)random numbers consumed at low-end edge devices increases, whereas crypto-secure random generation remains unaffected by this paradigmatic shift. Conversely, new AI platform architectures may counter these increased demands.

We evaluate different pseudo-random number generators (cf., Section 9.6) on two single-core RISC-V based processors of the PULP family and compare the performance to an off-the-shelf ARM Cortex-M core. Table 9.9 summarizes the hardware properties of our reference platform: The VEGAboard [293] holds multiple cores on the RV32M1 chip, which it can operate independently. We use this platform to compare ARM and RISC-V properties. In terms of processor complexity and application targets, the ZERO-RISCY [310] is on par with a Cortex-M0+ processor, while RI5CY [311] is on par with Cortex-M4. All four cores operate at the same CPU frequency and share memory as well as peripherals which includes a TRNG.

Table 9.10: Throughput and energy consumption per integer (#) for non-crypto PRNGs running on three different processor cores of the VEGAboard.

| | Byte rate [kB/s] | | | Average energy per # [nJ][a] | | |
|---|---|---|---|---|---|---|
| Generator | ZERO-RISCY | RI5CY | Cortex-M4 | ZERO-RISCY | RI5CY | Cortex-M4 |
| Mersenne Twister | 1118 | 1527 | 2290 | 79.5 | 60.3 | 43.3 |
| Tiny Mers. Twist. | 2158 | 2935 | 2608 | 49.3 | 36.7 | 39.4 |
| Xorshift | 5869 | 6264 | 7227 | 14.8 | 13.2 | 10.9 |
| Minimal Standard | 1715 | 2331 | 2134 | 61.9 | 46.9 | 50.9 |
| Knuth LCG | 3682 | 4697 | 4581 | 26.3 | 20.2 | 20.2 |
| TRNG | $2.4{\cdot}10^{-2}$ | | | $2.1{\cdot}10^{6}$ | | |

[a]Standard deviations are low for PRNGs ($\sigma < 1.5\,\%$) and high for the TNRG ($\sigma \approx 22\,\%$).

Table 9.10 presents our measurement results running five non-crypto PRNGs on three cores of the VEGAboard platform, as well as the embedded TRNG. The pseudo-random number throughput ranges from 1000–6000 kB/s on ZERO-RISCY, and RI5CY speeds up by a factor of 1.1–1.4 which is achieved by additional hardware extensions of the processor. The Cortex-M4 increases performance of PRNGs that involve multiple XOR operations (Xorshift, Mersenne Twister), though, RI5CY outperforms the Cortex-M4 by 10 % for the others, making effect of the ALU extensions and hardware loops.

Energy demands reflect similar properties with $\approx 20$ nJ consumption on both RI5CY and Cortex M4 while operating the recommended Knuth LCG. The fastest PRNG Xorshift is content with 10 nJ per integer on the Cortex-M4, which is only 75 % of the RI5CY consumption. The TRNG performance operates at the lower end compared to other hardware generated random numbers (Section 9.5.3), however, it should be involved only for seeding.

Despite small variations, the performance of random number generation on a single core scales similarly to our measurements presented in Section 9.6.6, regardless of the processor architecture. Our measurements are also in rough agreement with the single-core results presented by Wang *et al.* [393] who found a speedup factor of 1.3 for RI5CY over Cortex-M, operating neural networks. For multi-core processors, we expect the same upscale to hold and $\approx 6$ x speedup with about 50–70 % energy savings for pseudo-random number generation. In the context of machine learning at the edge, random input can be parallelized with these concurrent hardware architectures, which addresses high processing requirements at low energy consumption.

As an alternative, Forooghifar *et al.* [112] introduce "self-awareness" as an architecture-independent solution to improve nodal lifetime when operating machine learning at the edge. Complex computations are outsourced from battery driven nodes and distributed between the edge, fog, and cloud, following an energy estimation on the constrained node.

Processing demands for AI are and will continue to be costly on IoT edge devices. In contrast,

many simple sensor nodes will commonly process only little data and require only a single processor. Consequently, performance enhancements of a targeted AI hardware platform will enfold limited impact in conventional IoT use cases, as indicated by our measurement results.

## 9.8 Discussion: Hardware or Software for Randomness in the IoT

An increasing number of embedded controllers is expected to provide hardware primitives for basic cryptographic operations in the near future, which bring promise of fast and efficient random generators and contribute real entropy. The performance of on-chip hardware based random number generators, however, is heterogeneous. Some devices operate fast and save battery resources (*i.e.,* STM32F4), while others are slow and require notably larger amounts of energy (*i.e.,* nRF52840) than corresponding software. As shown in Section 9.5, few devices even produce poor statistical output (*i.e.,* CC2538). Some manufacturers advice against an immediate use for cryptographic random number. Other manufacturers describe their on-chip random number generators as suitable for cryptographic purposes without providing a cryptographic proof (Nordic). Truly random generators rely on real entropy from a physical process. This may be influenced by environmental factors, which opens an attack surface compared to seeded pseudo-random number generators.

Common IoT operating system such as RIOT need to make decisions on which hardware functions to include and how to integrate hardware and software components into the random subsystem. These multi-platform multi-purpose systems want to provide an overall lean solution of reliable quality at a predictable performance. To aid this design process, we now analyze key performance properties of the different hardware- and software-based random number generators that are either unseeded (TRNG), seeded ((CS)PRNG), or hybrid.

In Figure 9.10, we compare energy consumption and average run-time per integer (left) as well as the average current flow (right) for the hardware systems summarized in Table 9.2. Software PRNGs are measured on the STM32F4 board. We observe that the lightweight software generators are fastest and consume the least energy together with the TRNG on the chip of STM32F4. While software PRNGs operate up to five times faster, they charge a higher current than the TRNG which results in a similar energy footprint. On-chip hardware generators on the CC2328 (HWPRNG) and MKW22D (TRNG) devices consume about one order of magnitude more time and energy. Hence, we argue for the choice of a lightweight software generator (*i.e.,* the Knuth LCG) for the regular production of random numbers. Hardware generators are best used for (re-)seeding, when true entropy is required.

Running a cryptographically secure generator demands unsurprisingly more energy resources in comparison to its lightweight PRNG alternatives or the STM32F4 on-chip TRNG. Among the three software CSPRNGs, SHA256PRNG is clearly most frugal with an energy consumption of $0.5\,\mu$J for one integer and an average current consumption of 17 mA. Note that caching is

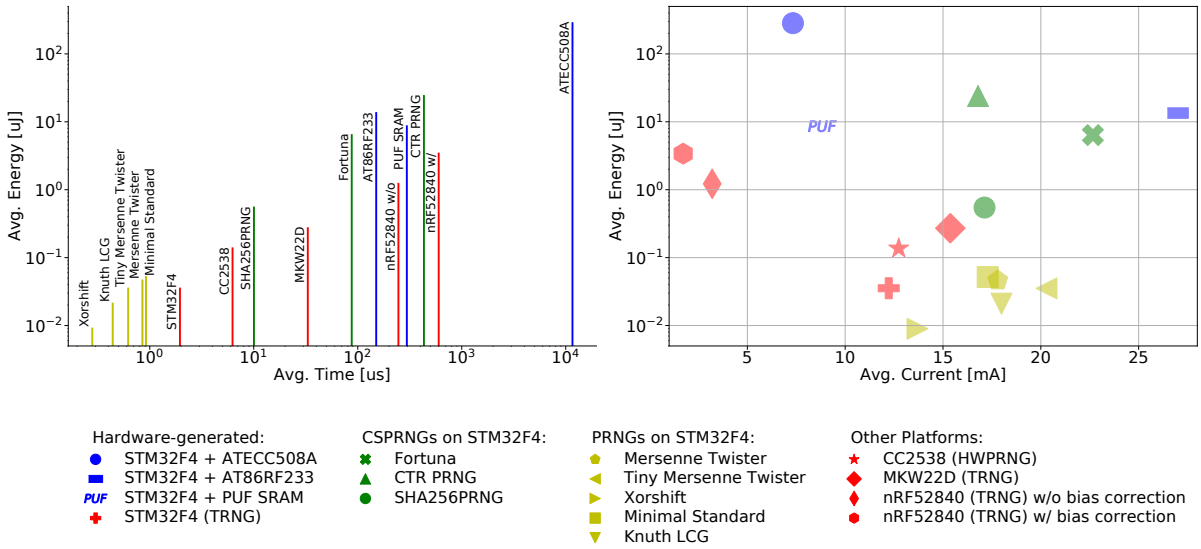



Figure 9.10: Average energy consumption over average time (left) and average current draw (right) for hardware and software generated random integers.

involved here, thus, only every eighth requested integer involves computation of a new hash which then causes currents up to 20 mA. Fortuna and CTR PRNG drain notably more energy in comparison to the SHA256PRNG as already visible in the previous analysis. CTR PRNG consumes up to 24 $\mu$J for one integer, which is four times more than Fortuna, even though its average current consumption is similar to the SHA256PRNG that uses 5 mA less than Fortuna. The increased energy consumption for one integer may be compensated in a stream or by caching intermediate numbers. Hardware based random numbers on the nRF52840 controller consume energy similar to the CSPRNGs, although the low-power MCU drains less than 5 mA in both operation modes (w/ and w/o bias correction). This is due to its slow operation.

The external ATECC508A chip, which implements an approved HWCSPRNG with seeding in hardware, exhibits a worse performance than all software CSPRNGs. The throughput of the software solutions is 20–100 times higher while the energy consumption remains 3–10 times less. The average current of the ATECC508A solution remains small in comparison to the software solutions that do not have to power a second device. This is because the main controller idles while the external chip processes random values. The external chip requires only a small current as shown in Section 9.5.3. Eventually, the total energy consumption can be improved by further driver optimizations. A minimum of 280 $\mu$J is required to request from one up to eight integers. It is noteworthy that this chip implements secure seeding on its own, which enhances very constrained devices without proper entropy sources. Furthermore, it implements tamper detection in hardware which provides additional protection against side channel attacks with physical device access.

Memory limitations of very constrained devices (*e.g.,* ATmega2560 with only 8 kB RAM

and 256 kB Flash) are unable to run complex software CSPRNGs due to memory constraints. The Fortuna CSPRNG requires almost 50% of the ATmega2560 RAM leaving only 4 kB for firmware and the remaining security protocols, which is insufficient for real-world IoT-networking applications [34]. Instead, the device driver for the ATECC508A chip can be included which (i) reduces memory requirements of cryptographic functions, (ii) offloads processing of complex algorithms on the device, and (iii) includes additional crypto-related features.

Next to the dedicated crypto-chip, other external randomness generators, namely the AT86RF233 transceiver as well as the SRAM PUF are energy expensive using up to one order of magnitude more than the SHA256PRNG. Both mechanisms, though, are not designed to be used periodically. Instead, they act as seed sources that are utilized once during instantiation of a PRNG and on re-seeding. The current consumption differs notably between the transceiver and the PUF SRAM. The transceiver powers two devices in active mode because the MCU polls random bytes via the SPI while the radio needs to stay in receive mode. This draws a current of up to 26 mA in total. The PUF SRAM on the other hand takes twice as long to retrieve a high entropy integer but it drains less than 10 mA on average. As depicted in Section 9.5.1, this procedure needs to take place very early on startup—even before clock and bus initialization in the operating system.

In summary, we argue that seeded pseudo-random number generators—a lightweight general purpose PRNG or an approved CSPRNG—are the preferable solution for producing (secure) random numbers. Co-processors and external hardware assistance are vital for adding entropy and can help to reduce memory footprints for tiny devices. They lead to a decreased throughput, though, when compared to software that runs on the main controller. Conversely, special crypto-chips can offload processing demands from the main controller. Transceivers as well as uninitialized SRAM are essential on devices lacking TRNGs. Even though PUF SRAM and the radio-based entropy sources are costly, they uniquely contribute entropy and are rarely needed for seeding purposes.

## 9.9 Conclusions

In this chapter, we explored the building blocks for randomness in the constrained Internet of Things: hardware and software components that generate statistical randomness, entropy, and resilience against cryptographic attacks. We systematically derived the requirements for IoT random subsystems from the perspectives of statistics, security, and operating system integration. An extensive, comparative evaluation using several prominent test suites as well as detailed performance measurements on popular devices delivered insights into the overall quality and suitability of the different components under test. This work derives four major recommendations:

1. Separate general purpose random generators from cryptographically secure generators on the OS level. Avoid any mixture or interference between the two.

2. Prefer (software) PRNGs over random generating hardware, as they are more efficient and reliable. Exploit hardware components as additional entropy sources for (re-)seeding or when CSPRNG operation is infeasible on a constrained node.

3. The Knuth LCG is the most efficient general purpose generator that provides decent statistical quality. It is simple and lean enough to run on very constrained devices.

4. We recommend SHA256PRNG as a cryptographically secure generator, since it outperforms its competitors by an order of magnitude in several dimensions.

With this work, we hope to contribute to a thoughtful development toward a secure Internet of Things. This will be of particular importance, as more and more (sensitive) data originates from IoT nodes and needs protection. Content object security with OSCORE [335, 150] and LAKE [385], for example, will facilitate the encryption of individual information units, but will extend the use of cryptographic primitives such as random numbers during operation.

# Chapter 10

# Seed- and Key Generation with Physical Unclonable Functions


## Abstract

Security is essential for the Internet of Things (IoT). Cryptographic operations for authentication and encryption commonly rely on random input of high entropy and secure, tamper-resistant identities, which are difficult to obtain on constrained embedded devices. In this chapter, we design and analyze a generic integration of physically unclonable functions (PUFs) into the IoT operating system RIOT that supports about 250 platforms. Our approach leverages uninitialized SRAM to act as the digital fingerprint for heterogeneous devices. We ground our design on an extensive study of PUF performance in the wild, which involves SRAM measurements on more than 700 IoT nodes that aged naturally in the real-world. We quantify static SRAM bias, as well as the aging effects of devices and incorporate the results in our system. This work closes a previously identified gap of missing statistically significant sample sizes for testing the unpredictability of PUFs. Our experiments on COTS devices of 64 kB SRAM indicate that secure random seeds derived from the SRAM PUF provide 256 Bits-, and device unique keys provide more than 128 Bits of security. In a practical security assessment we show that SRAM PUFs resist moderate attack scenarios, which greatly improves the security of low-end IoT devices.


## 10.1 Problem Statement and Related Work

Pappu *et al.* [297] are the first to introduce "physical one-way functions" and the notion of a PUF dates back to Gassend *et al.* [121]; both describe a technique to uniquely identify and authenticate individual integrated circuits. The research community identified PUFs as an attractive solution for the IoT [323], because the intrinsic hardware variations can feed security primitives on low-end devices without increasing hardware cost. Orthogonal to PUFs that utilize variations of low-cost, multi-purpose building blocks, research also advances in the field of hardware security at the transistor level [324, 360, 169]. PUFs can be distinguished into two classes [297, 138]. They either process many inputs (*i.e.,* challenges) to produce varying outputs

(*i.e.,* responses), or only few inputs which produce few, or only one response. More precisely, a PUF is denoted as strong if it provides a large challenge-response space, whereas a weak PUF provides only few challenge-response pairs that typically scale linearly or polynomially with the design size [260]. Hence, a memory readout which produces one response can be classified as a weak PUF.

A variety of use cases for PUFs emerged, such as secure key storages [100], communication protocols [68, 49], supply chain security [105], remote attestation [333], firmware updates [308], or generic trust anchors [255].

The security of these applications as derived from PUFs is only as strong, as the secrets extracted from the underlying hardware variations. In this section, we review the fundamental properties, basic assessment measures, and pose the question of potential weaknesses in PUFs.

### 10.1.1 Properties of Uninitialized SRAM

Reading out uninitialized SRAM produces a digital fingerprint. Manufacturing processes introduce variations in the silicon of transistors that construct a memory cell. When powered on, some cells drift to the logical state 1, others to 0, and cells without bias fluctuate according to environmental conditions. The resulting patterns require a careful assessment between devices (inter-device) in order to estimate their uniqueness, and between power-cycles on one device (intra-device), in order to quantify the (random) noise. This noise can be utilized as an entropy source, or needs to be removed for reliably reproducing an exact version of the pattern.

**Aging Bias.** Uniformly random variations result in an equal proportion of stable cells that power up with 0 or 1 on a single SRAM pattern. Aging and utilization, however, skew this distribution, due to drifting voltage threshold values of the transistors that form a memory cell [106]. The increased probability for one symbol (1 or 0) introduces a bias, which in turn benefits an attacker, who tries to guess bit values. Guin *et al.* [139] find a bias of up to 54 % under artificial aging. Holcomb *et al.* [173] counter that 'normal' use patterns of intermittently powered devices prevent an identical skew.

The state-of-the-art motivates additional and more realistic analyses of bias and aging effects on platforms that naturally aged while executing real-world IoT applications over a long period of time (see Section 10.3).

**Static Bias.** In contrast to an aging bias, real-world PUFs may be affected by a static bias at certain bit positions [398]. Rahman *et al.* [312] observe systematic correlation between SRAM patterns across chips, and cell-neighborhood interactions due a systematic physical arrangement on the silicon. Both effects reduce the device uniqueness. An attacker, who owns a chip of the same type, could utilize a local measurement to guess bit values at specific positions with a better chance than 50 %, which facilitates prediction of a secret value derived thereof.

Conditioning the SRAM resolves systematic bias. Bit selection [404, 209] is an approach to exclude biased bit addresses of the SRAM, but adds enrollment complexity for each individual

node. Storing a bit mask for cell selection requires additional memory, which conflicts with limited memory resources on IoT devices. Instead, extending the SRAM PUF input increases the total amount of unbiased bits that generate a secret, which ideally prevents successful guesswork. Increasing the length, though, threatens *fuzzy extraction* (see Section 10.1.4) which may leak information about the secret, and requires a careful assessment of (*i*) the remaining entropy as well as (*ii*) the increase in processing overhead on resource-constrained nodes.

**Environmental Bias.** PUFs are subject to noise, which is affected by environmental operating conditions. Related work analyzes SRAM startup patterns under varying voltages and temperatures. A body of work shows that SRAM PUFs are robust against variations of the supply voltage [332, 185, 74, 38], tested at $\pm 10\,\%$ of the nominal value. Adjusting the voltage ramp-up speed [79, 228] can mitigate effects of temperature variations, which affect the SRAM startup behavior more severely. These solutions require special hardware, though.

The effect of temperature variations on the startup pattern of SRAM is commonly analyzed within the industrial operating temperature range of -40°C to +80°C. Leest *et al.* [375] quantify the **intra-device** relation between startup pattern of a device, which shows that the min. entropy is minimal at a low temperature of -40°C, and gains up to $\approx 2\,\%$ of noisy cells when the temperature increases to +80°C. When quantifying the required SRAM length for seed generation, the lower bound should serve as a conservative starting point, while higher temperatures improve seed generation due to higher entropy.

Schrijen *et al.* [332] compare the intra-device hamming distance of SRAM patterns taken under different temperatures, compared to an enrollment readout at ambient temperature. The startup noise between patterns of the same device almost doubles when increasing the operation temperature from 20℃ to 80℃. Claes *et al.* [74] present an increase of the fractional hamming distance from 0.06 to 0.1 at the extreme operating temperatures of -40/+80°C. Katzenbeisser *et al.* [185] present similar results but find that the SRAM PUF is more robust against varying operational conditions compared to other PUFs (*e.g.,* arbiter or flip-flop PUFs). Overdesigning the error correction code can mitigate this effect, but increases the computational complexity—sometimes in conflict with IoT device constraints as well as the remaining key entropy.

Holcomb *et al.* [173] show that the increase in noise which is introduced by temperature variations overrules a predictable aging effect of NBTI. This has a positive effect on the **inter-device** uniqueness, which increases with the absence of an identical skew.

### 10.1.2 Empirical Evaluation of PUFs

The common measure to quantify the unpredictability of a pattern is given by the min. entropy metric:

$$H_{min}(p_{max}) = -log_2(p_{max}) \tag{10.1}$$

For a single bit, $p_{max} = max(p, 1-p)$, *i.e.,* the maximum probability for attaining one ($p$) or

zero $(1-p)$ at the same SRAM bit position. An ideal probability of $p_{max} = 0.5$ maximizes the min. entropy to $H_{min} = 1$. This metric is used to assess intra-device variations across multiple pattern of the same device, or inter-device variations between the pattern of multiple devices. Random noise increases the intra-device min. entropy after a power-cycle, which facilitates seed generation, but challenges a reliable key construction. Over-dimensioning the *fuzzy extractor* (see Section 10.1.3) can mitigate this effect, but increases the computational complexity—in conflict with IoT device constraints.

Schrijen *et al.* [332] present intra-device measurements across SRAM technologies in differing setups and find that variations across SRAM of different vendors are not significant. Katzenbeisser *et al.* [185] show that the inter-device min. entropy is invariant to temperature. This enables longer repetition codes to correct multiple errors, which would otherwise leak secret information in the case of a low inter-device entropy (*cf.* Section 10.6).

The inter-device min. entropy assesses device uniqueness and the impact of bias. The literature reports inter-device min. entropy values from 0.7 [74] to 0.9 [206] between SRAM patterns. Quantifying this metric requires multiple samples which is particularly challenging since it involves many nodes.

**Min. Entropy Convergence.** The maximum probability $p_{max}$ in Equation 10.1 can be empirically sampled from a limited number of probes $n$ (*i.e.,* nodes). Then the empirical estimator

$$H'_{min}(i,n) = -log_2 \left[ max\left(\frac{i}{n}, 1-\frac{i}{n}\right)\right] \tag{10.2}$$

with $i$ positive events (ones) in $n$ samples from individual nodes converges to the min. entropy in Equation 10.1. Statistical convergence, however, is slow. According to the central limit theorem [108],

$$\left|H_{min}(p_{max}) - H'_{min}(i,n)\right| \sim \frac{\sigma}{\sqrt{n}} \quad (\text{as } n \to \infty), \tag{10.3}$$

where the dispersion $\sigma = \sigma_{H'_{min}} \approx 1$.

Hence, estimating the inter-device bias from 100 samples of SRAM PUFs still includes an error of 10%. Accordingly, the largest available SRAM evaluation of 144 nodes [396] bears an uncertainty of more than 8% This shows the need for significantly larger samples in order to approximate the inter-device min. entropy accurately, which we will present in Section 10.3.

**Bit-Aliasing.** Maiti *et al.* [251] introduce *bit-aliasing* to quantify systematic inter-device bias (cf. Section 10.1.1) among 125 FPGAs that implement a ring oscillator (RO) PUF. Large-scale evaluations of RO PUFs on 217 FPGAs [171, 137], and 133 ASICs [408] show that the location of cells within the FPGA affect performance properties. The bit-alias of uninitialized SRAM between 50 [38] and 144 [396] devices reveals a slight double-peaked distribution of the bit-alias scores due to SRAM layout systematics, but seem to miss convergence due to an insufficient sample size.Wilde *et al.* [397] identify a research gap in convergence and deduce that qualified

inter-device bit-alias measurements require more than 600 devices to converge with an error below 5%.

Quantifying possible inter-device correlations using hundreds of devices demands for high cost and engineering efforts. In the subsequent analyses, we will tackle these challenges by taking advantage of a large-scale testbed.

### 10.1.3 Random Seed and Key Generation

SRAM PUFs promise to support bootstrapping security on embedded IoT nodes by deriving random seeds and private keys from uninitialized memory. Commercial IoT platforms more and more provide isolated PUF circuits for this purpose, but an open software implementation that enables PUF-functionality without dedicated PUF-circuitry is missing. To enable software-based SRAM PUFs on a wide range of heterogeneous IoT platforms, the hardware abstraction layer of an operating system can enable low-level hardware access and facilitate PUF-based seed and key generation.

**Seed Generation.** Random numbers are essential for security. Commonly, a sequence generated by a true random number generator (TRNG) acts as seed or refresh value for a pseudo-random number generator (PRNG) as well as a cryptographically secure PRNG (CSPRNG). Van der Leest *et al.* [375] derive the min. entropy of repeated SRAM startup patterns on a device for creating a random seed value. The concept was applied to off-the-shelf MCUs [376] and revealed a diverse picture. Not all embedded SRAM technologies are qualified to produce high entropy seeds. SRAM, so the lessons learned from this study, must be analyzed prior to deployment. Krentz *et al.* [210] propose an SRAM seeding mechanism and add antenna noise to uninitialized memory pattern. The combined values are conditioned with a van Neumann extractor, which introduces variable runtime overhead.

SRAM must be uninitialized to obtain entropy between power-cycles, which is why the PUF operation should only take place during system startup before the memory has been utilized. This startup sequence, however, might be executed without a cold boot, possibly leading to zero-entropy seeding. Hence, a PUF implementation needs to ensure a preceding power-off cycle.

**Key Generation.** A reliable key generation depends on the removal of random noise. The related concept of *fuzzy extraction* was first presented in the context of biometric authentication systems [182, 90] to reliably reconstruct an exact version of a reference measurement. Fuzzy extractors are based on error correction codes. Error correction schemes for PUFs [407] were evaluated on an FPGA [172]. For complexity reasons, not all codes are applicable to low-end devices with its constrained resources. Korenda *et al.* [209] reduce the computational requirements by identifying stable values before encoding, which reduces the error probability. Leest *et al.* [374] propose specific hardware implementations for soft-decision decoders, which

improve the correction capabilities and require only half of the PUF bits for secret generation compared to hard decision decoders.

A deployment of a fuzzy extractor proceeds in two phases, *enrollment* and *reconstruction*. The enrollment is a trusted process and produces *helper data* [86], which is later used to reconstruct the PUF value. Helper data is publicly stored in non-volatile memory.

A PUF response does not contain maximum entropy, which flaws its immediate use as a cryptographic key. Besides, it may be too long or too short, adjusting its length to include a required amount of entropy. For mitigation, a compression scheme can be used to create a key with maximized entropy and to preserve forward secrecy of the PUF response. Practical implementations [99, 59, 249] employ a cryptographic hash function that compresses the lengthy PUF response.

Error correction [86], and crypto-processing [196] quickly exceed the computational-, and energy resources on constrained embedded devices. A modular and configurable PUF implementation should ease the deployment under varying environmental conditions and adjust to the capabilities of heterogeneous platforms (*e.g.,* processing power, availability of crypto-acceleration).

### 10.1.4 Security Analysis of PUFs

The related work presents threats to PUFs mainly from three angles.

**Analytical Attacks.** Public helper data techniques leak information if the PUF is biased [409]. For the *code offset* method, helper data lengths should be kept small to avoid information disclosure–in conflict with PUF bias which may increase the required length. Koeberl *et al.* [206] conservatively estimate the entropy loss during helper data construction for varying error correction codes in the fuzzy extractor, but were criticized to be overly pessimistic [86]. Maes *et al.* [250] present methods that calculate the entropy leakage exactly, and de-biasing which resolves bias on an FPGA. Liu *et al.* [242] present countermeasures to bias on an MCU.

**Modeling Attacks.** PUFs are susceptible to modeling attacks [119, 344, 400]. Rührmair *et al.* [320] apply machine learning to challenge-response pairs of PUFs with many inputs and predict their outputs, which requires the ability to eavesdrop PUF responses. Strieder *et al.* [357] exploit helper data of PUFs with many inputs for training. PUFs with few (or only one) input are less vulnerable to learning attacks due to restricted input/output variables.

**Hardware (Invasive) Attacks.** Helfmeier *et al.* [170] cloned SRAM of a common IoT device using a focused ion beam instrument. Zeitouni *et al.* [411] present a side-channel analysis on an SRAM PUF, using remanence decay. Both attacks require physical control of the device under attack.

These analyses are tied to specific algorithms, or dedicated PUF implementations in hardware or software. A practical threat model that analyses the remaining security risks of SRAM PUFs on low-end hardware from the perspective of an IoT operating system is missing. We will fill this gap in Section 10.7.

## 10.2 Experimental Setup

We want to analyze the properties of uninitialized SRAM on a large scale, and assess our measurements on IoT-typical constrained hardware. Therefore, we chose an existing testbed as an evaluation environment (Section 10.2.1), which provides many off-the-shelf nodes (Section 10.2.2) and grants open (remote) access for reproducibility. The drawback of this approach, however, is that we cannot vary the operational conditions of nodes.

On the software side of our experiments (Section 10.2.3), we chose the open source IoT operating system RIOT [34] for three reasons. (*i*) RIOT is an off-the-shelf OS that is used in many IoT deployments [96] with support of numerous heterogeneous platforms. In RIOT, PUF support brings benefit to a broad range of systems and applications. (*ii*) It provides support for the FIT IoT-LAB testbed nodes. This allows us to easily benefit from the existing tools and facilities. (*iii*) An active open source community, which had first hand experiences with initial SRAM PUF trials [194], facilitates code contributions.

### 10.2.1 Testbed Environment

We conduct our experiments on the FIT IoT-LAB testbed [5] to attain a large number of nodes. The testbed consists of seven sites with different topologies and a total number of more than 1500 nodes of 25 architectures. The *M3* nodes make up the majority ($\approx$ 800 nodes) and reflect properties of commercial off-the-shelf class 2 IoT devices [58]. Each node is attached to a control node which provides a power monitor (INA220), allowing to measure the operational voltage and current that flows to the MCU and the external board components. Nodes are deployed across facilities of INRIA in France. Hence, all our experiments are conducted under environmental conditions of work offices.

We use 708 *M3* nodes in our experiments. To automate experiment control, we utilize the command-line interface `iotlabcli`. Nodes serial outputs are piped to individual log files. Note, when reproducing the experiments, high data volumes are generated, while testbed users have limited disk quota. Data compression, moving files periodically, and asking for increased quota can assist.

### 10.2.2 Hardware Platform

**Testbed.** *M3* nodes consist of a 32-bit ARM Cortex-M3 CPU, integrated into the STM32F103REY MCU, which runs at max. 72 MHz and provides 64 kB embedded SRAM, and 512 kB internal flash. The MCU offers common features that we exploit: (*i*) Low-power standby mode turns off the whole SRAM. All content in SRAM and registers are lost, except for the backup domain. (*ii*) Real-time clock remains operable during standby, to trigger an interrupt for wakeup. (*iii*) Power control registers indicate whether the MCU has been in standby after a system restart. But this MCU lacks hardware security features, *i.e.,* a random number generator, crypto-accelerator,

and secure key storage. *M3* nodes additionally connect external components via SPI: An 16 MB external NOR flash allows storing data persistently, and a low-power radio which is the only alternative to gather entropy on this board, by sampling antenna noise.

The microchips of the *M3* nodes in the FIT IoT-LAB testbed originate from two lots and four wafers, two of which build the majority of devices. We conducted several experiments to find a systematic variation. But we could not find significant differences between these batches, hence, we treat them equally in our evaluation and exclude the results of the batch comparisons.

**Local.** To evaluate the PUF performance on heterogeneous IoT devices with varying architectures, we also perform local experiments on two different off-the-shelf IoT platforms, and measure the processing time on a single device per platform: The *ESP32*, which consists of an Xtensa 32-bit CPU with 520 kB SRAM, 4 MB flash, and operates at max. 240 MHz. The *HiFive* which consists of a RISC-V RV32IMAC CPU that provides 16 kB SRAM, 4 MB off-chip flash, and operates at max. 320 MHz.

### 10.2.3 Software Platform

We base our PUF implementation on RIOT 2022.01. It supports different architectures (8–32-bit CPUs), over 150 MCUs, and nearly 250 IoT boards. The OS provides multi-threading with preemption, power management, and a hardware abstraction layer to enable portability. We utilize and complement these features in our implementation (Section 10.4). RIOT provides its own IPv6 network stack (`GNRC`) and supports multiple low-power radios as well as wired interfaces. For the *M3* nodes, we added drivers to access power control registers and the external flash memory. To broaden our experimental basis, we integrated the PUF initialization to the *ESP32* and *HiFive* architectures.

In our experiments, we trigger repeated power cycles on the nodes. For this, we utilize existing RIOT interfaces, namely, the power management (PM) interface to enter standby, which turns-off the SRAM, and the real-time clock (RTC) to generate a future wakeup interrupt.

## 10.3 Large Field Study of Uninitialized SRAM

### 10.3.1 Inter-device Correlation

We want to analyze the similarity between individual SRAM patterns. Therefor we read the whole memory of 708 available *M3* nodes and compute the Pearson product-moment correlation coefficient which is defined as:

$$r(a,b) = \frac{\sum_{i=1}^{m}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{m}(a_i - \bar{a})^2}\sqrt{\sum_{i=1}^{m}(b_i - \bar{b})^2}} \tag{10.4}$$

where $a$ and $b$ denote the SRAM pattern of two devices with a length of $m$=64 kB. Figure 10.1 presents the matrix of correlation coefficients between the SRAM readout of all node pairs, as

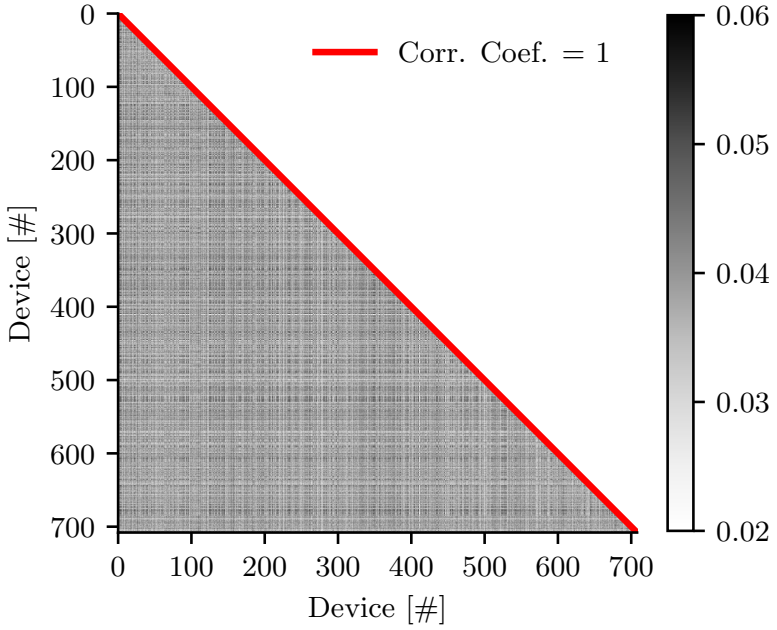



Figure 10.1: SRAM correlation between 708 nodes. The Pearson product-moment correlation coefficient of each pair is encoded in gray intensity. Autocorrelation results in a coefficient of one.

a measure of linear dependency between nodes. A coefficient of 1 indicates perfect correlation (pairs are equal), -1 represents negative correlation (pairs are opposite), and 0 means (linear) independence. All coefficients are small with a small positive bias (0.02–0.06), which indicates high independence between the memory patterns and motivates their usage as PUF source. Certain samples, however, indicate a slightly increased coefficient when compared to others. To better understand these correlations, we chose to further analyze the inter-device relations with a metric that incorporates the bit locality, *e.g.,* the bit-alias [251] quantifies inter-device bias (*cf.* Section 10.3.2).

### 10.3.2 Analysis of Static Bias

We calculate the bit-alias which quantifies the proportion of zeros and ones at every bit position $j$ in the memory pattern between $n$ devices:

$$\hat{p}_j = \frac{1}{n} \sum_{i=1}^{n} p_{j,i}, \tag{10.5}$$

where $p_{j,i}$ denotes the measured bit probability at position $j$ of device $i$. If the probability for attaining one or zero is unbiased, the expectation value at each bit position equals 0.5 and the distribution of the empirical $\hat{p}_j$-values follows a normal (error) distribution. Our evaluation

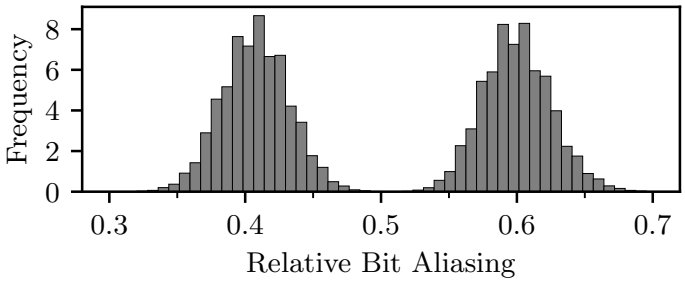



Figure 10.2: Distribution of bit-alias values between 708 nodes.

indicates repetitive pattern with (multiples) of 32 Bit blocks. Rahman *et al.* [312] find similar effects and relate this to the physical layout of the SRAM. Figure 10.2 displays the histogram of the bit-alias metric for all bit positions of the 64 kB memory. It reveals a bimodal distribution with peaks around 0.4 and 0.6. Previous work [396] suggested a double-peak distribution, but its sample size was too small [397]. To the best of our knowledge, our results show the first SRAM evaluation of the bit-alias with an error around 3 %.

Inter-device correlations in regions of SRAM can be beneficial for an attacker. Analysing a large set of equally produced devices may assist prediction of SRAM bit values at certain positions. In detail, a deviation of $\approx 0.1$ from the ideal bit probability of $p_j = 0.5$ increases the chance of guessing the correct value by 10 %. This lowers the inter-device entropy, which we quantify in Section 10.5.1, and requires careful consideration when generating keys (see Section 10.6.2). While not all SRAM technologies seem to be affected by this inter-device correlation, a preselection of uncorrelated bits for the enrollment process can mitigate this effect [312].

### 10.3.3 Analysis of Aging

The MCU age is noted on the chip package and our local *M3* sample devices indicate a production date in January 2012. This is in line with testbed statistics that date back the first experiment to September 2012. We further managed to get experiment metadata from the testbed team. Figure 10.3 displays the active utilization time of our test nodes since their deployment until the end of 2021. The majority of our publicly accessible nodes have been operated 2.5–8 thousand hours since their deployment. Thus, in contrast to prior work, we analyze devices that naturally aged under real-world conditions.

We want to analyze whether certain devices or memory blocks show anomalous behavior caused by aging or wear-out from similar firmware images. To that end, we quantify the intra-device bias by calculating the relative hamming weight:

$$HW(r) = |\{r_i \neq 0 : 1 \leq i \leq m\}| \cdot \frac{1}{m} \tag{10.6}$$

where $r$ denotes the bit value of one device at position $i$ in a block of the length $m$. Hence, the

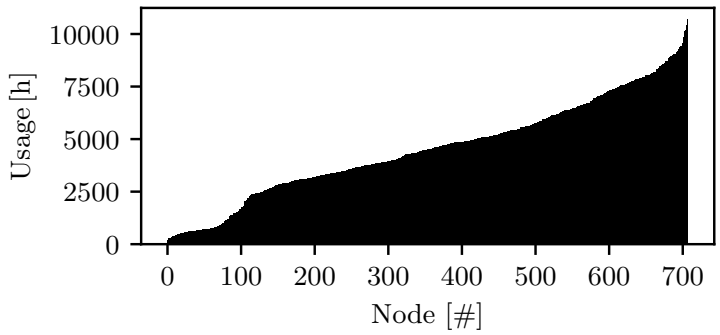


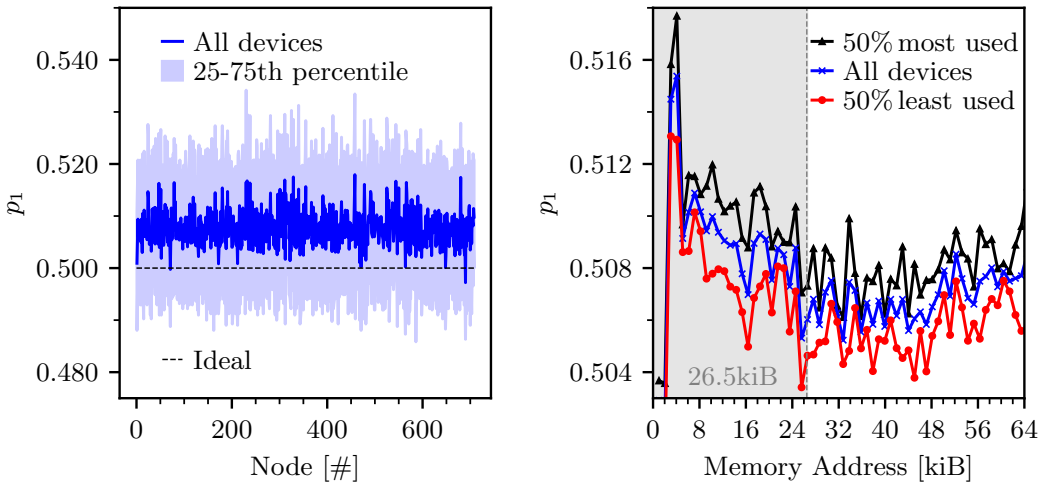

Figure 10.3: *M3* node active experiment operation time in hours. Nodes are ranked according to their utilization.

hamming weight reflects the proportion of ones ($p_1$). The proportion of ones and zeros should be equal ($p_1 = 1 - p_0 = 0.5$) without bias. Figure 10.4a displays the intra-device measurements across the whole memory of all boards ($m$=64 kB) which shows an average hamming weight of $0.508 \pm 0.003(\sigma)$. This slight (positive) bias is the effect of aging and is still small compared to the results of Guin *et al.* [139] who find biases of up to 0.54 after 336 hours (14 days) of stressed operation.

Figure 10.4b displays the hamming weight separated into memory blocks ($m$=1024 Bytes). We show average values across all devices, and two subsamples that include 50 % of the most and least used devices. (*i*) An increase at $\approx$4 kB is introduced by the bootloader. In real-world implementations, this can barely be avoided and PUFs should exclude SRAM at that region. (*ii*) The bias of heavier utilized devices increases less used ones by $\approx$0.0025, which confirms aging by operation, with a small magnitude. (*iii*) Besides (*ii*), the first $\approx$26.5 kB of memory exhibit a higher skew compared to the remaining. Common firmware sizes of large-scale networking experiments on these testbed nodes report (*e.g.,* [142]) memory requirements of 22–28 kB in RAM, which matches the region of systematic wear-out. Hence, we report strong indications of visible wear-out effect by long-term testbed utilization and avoid that memory region in our PUF design (Sections 10.4.4 and 10.4.5). Operating systems likely organize the program memory from the start of the address space. At the same time, real- world firmware images do not necessarily utilize the whole memory (uniformly), which fosters bespoke unbalanced wear-out effects. Testbed operators as well as PUF developers should include anti-aging techniques in the future to mitigate wear-out patterns of various characteristics in practice.

## 10.4 PUF Design for the RIOT OS

A wide availability of PUFs requires grounding in the ecosystem of an OS. The heterogeneity of supported platforms requires an integration into the configuration and the build system to adjust the diverse device properties. OS tests and tools provide useful interfaces to verify PUF




(a) Avg. and 25-75th perc. weight by device.

(b) Avg. weight by address.

Figure 10.4: 64 kB SRAM is split and analyzed in blocks of 1024 Bytes. The relative hamming weight is displayed for every device (10.4a) and memory address (10.4b); the latter distinguishes the half most/least used devices.

viability and to assess crucial configuration parameters (*e.g.,* required SRAM lengths). PUFs bootstrap system security and must therefore extend the OS startup code, module initialization, and finally the secure operation. Figure 10.5 presents an overview of our PUF integration in RIOT for creating (*i*) a simple seed for general purpose PRNG initialization, (*ii*) a secure seed for CSPRNG initialization, and (*iii*) a secret key. In addition, to ensure qualified PUFs, we provide a soft-reset detection mechanism that prevents initialized SRAM (*i.e.,* caused by insufficient power-off cycles) from generating seeds or keys.

### 10.4.1 Compile-time Configuration

RIOT supports many boards of largely varying hardware capabilities [55] that demand for a systematic compile-time modeling of its features. This modeling enables extensible code paths where possible, and facilitates reduced feature sets on platforms without certain hardware capabilities. RIOT uses a feature modeling based on Kconfig [366, 196]. Kconfig allows defining symbols that represent features, based on which dependencies and conditional default values are defined. For the PUF module, a platform can indicate *capabilities* as follows. `HAS_PM` enables low-power mode, and `HAS_PM_TIMER` enables programmatic wakeup from low-power mode. `HAS_PM_INDICATION` enables additional power-cycle detection during soft-reset detection. `HAS_CRYPTO_ACCEL` enables crypto hardware acceleration (future work).

Both seeders (Section 10.4.4) and the key generator (Section 10.4.5) provide *configurations* for PUF algorithms: (*i*) separate start addresses in SRAM, (*ii*) length of the considered SRAM

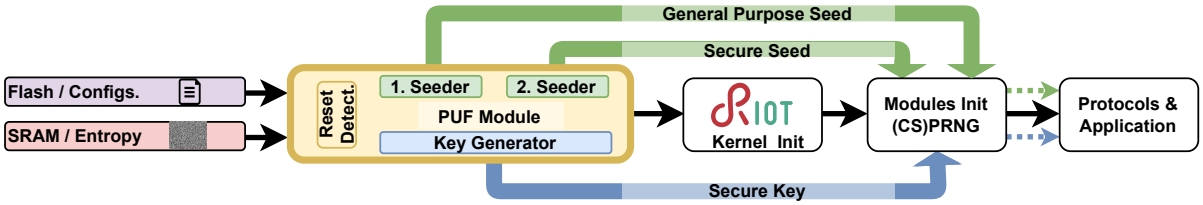



Figure 10.5: Integration of the SRAM PUF module in the IoT operating system RIOT.

blocks, (*iii*) choice of a cryptographic hash function,(*iv*) configuration of the error correction code for the key generator. Default values are chosen according to our evaluation.

### 10.4.2 Integration into OS Startup Routine

**System Reset.** RIOT provides a `reset_handler`, which is the start point after every system reset. A default startup routine follows four steps. (*i*) The data section is loaded from flash to RAM. (*ii*) The *.bss* (block starting symbol) section (used for uninitialized data) is set to zero. (*iii*) The MCU and board specific components are initialized. (*iv*) The OS kernel is loaded and (*v*) `auto_init` initializes modules prior to starting applications.We perform our PUF initialization prior to step (*i*), to obtain a pristine response of uninitialized memory.

**Linker Attributes and Erasure.** To prevent PUF outputs from erasure by the subsequent startup routine, a *.noinit* section in the linker script of every supported CPU architecture defines a PUF attribute with which we declare variables used to store the PUF seeds and keys. Seeds and the key are consumed during `auto_init` and unavailable by the end of the initialization (see Section 10.4.6).

**Startup Delay.** PUF execution adds a delay (see Section 10.6.3) to the system startup, which is primarily introduced by resource-intensive crypto-operations on constrained devices [196]. If available, crypto-accelerators can reduce that time. When operated in software, the execution may degrade due to the early PUF execution on perhaps uninitialized system clocks, prior to MCU initialization. An interface that allows conditional PUF execution during the next reset can mitigate this affect in the future.

### 10.4.3 Detection of Soft Resets

Memory must be uninitialized for PUF operations, which is achieved by a sufficiently long power-off cycle. Short resets can occur, however. In such cases, the PUF procedure must not be executed to prevent duplicate seeds and false key construction. Kietzmann *et al.* [199] present a simple detection mechanism to catch soft-resets. In a nutshell, the soft-reset writes a memory marker to a known address. On soft-reset, the marker will persist in memory. Conversely, a sufficient power-off cycle changes the value of the marker and enables PUF operation. One caveat of this approach is a false negative decision, which can be triggered by only a single bit flip

in the marker variable, while old values stay in memory. In a baseline experiment we analyzed the hamming distance between the marker variable and the expected value and decreased the duration of the low-power cycle. Our results show notable signs of memory retention (decrease of the hamming distance) below 3 ms, which should be excluded. Longer cycles, however, may still incur memory retention, sometimes in the order of seconds [332].

**Distance Detection.** A future soft-reset detection stage could improve the false negative sensitivity and incorporate a distance metric to the detection algorithm. Additionally, an individual marker per device could utilize the inverse startup pattern, maximizing the range of potentially flipping bits, which increases granularity.

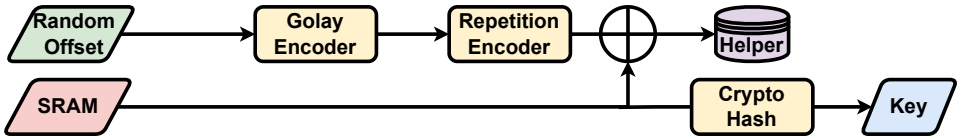
**Sleep State Report Interface.** The memory marker is at risk to be manipulated during runtime, by software defects, or intentionally by adversaries that manage to execute malicious code (Section 10.7). This can cause an undetected soft-reset, resulting in zero-entropy seeding and false key reconstruction. We extend the power management (PM) API in RIOT by a function to report the preceding state after a reset. Only if the marker-based detection fails *and* the system starts from deep sleep, PUF operation is executed. Not all platforms support this feature, unfortunately.

### 10.4.4 Random Seed Generation

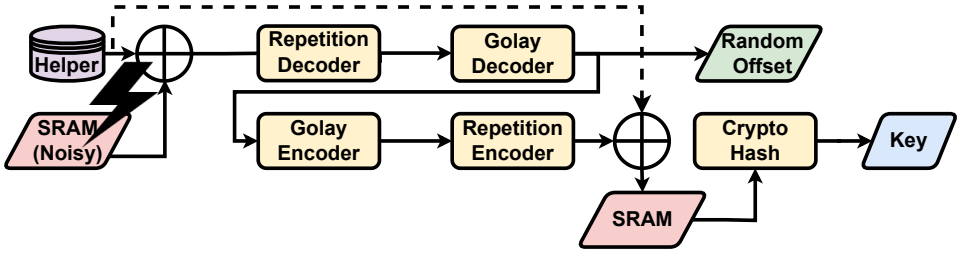
Uninitialized SRAM contains randomness for the seed generator and is compressed to provide a concise value of maximized entropy. We provision two seed generation functions that take as input the SRAM start address and considered memory length. By default, we utilize a randomly chosen start address in the center of the memory map, to circumvent systematic wear-out effects that likely occur in the beginning of the RAM (*cf.* Section 10.3.3), and we locate regions for both seed functions successively. Addresses and lengths can be configured, though. A dynamic mechanism could thus mitigate potential aging phenomena.

**Construction.** Our first seed is extracted by the lightweight DEK hash [205] and compressed to an integer value, which is utilized to seed a non-secure general purpose PRNG. The second seed is created for security purposes and bases on compression by a cryptographic hash (SHA256 by default). Hence, the size of the seed corresponds to the digest length. It can be utilized to feed an entropy accumulator, or the CSPRNG initialization directly. Potential CSPRNG re-seeding [199], however, requires a power-cycle to obtain fresh entropy from the SRAM.

**General Purpose vs Secure Seeds.** General purpose seeds must not be used in cryptographic contexts due to insufficient entropy and lack of forward secrecy. Conversely, cryptographic seeds can be used for general purpose, but exhibit higher cost (see Section 10.6.3). The same seed must not be used for both types of generators [199], since typical PRNGs are invertible, hence, their outputs disclose information about the initial value. Similarly to PRNGs, our general purpose seed generator is invertible. Consequently, this seed can disclose information about the




(a) Enrollment




(b) Reconstruction

Figure 10.6: A fuzzy extractor based on the code-offset construction. Offset is created at random. Deployments consist of enrollment, and reconstruction during regular device operation.

initial PUF response. Hence, cryptographic seed- and key generators should never operate on a memory region that was used by the simple seeder before.

**Secure Seeds on Soft Reset.** A fresh and secure seed that was used on CSPRNG initialization should be disguised after use to preserve privacy. Hence, we hash it after CSPRNG seeding and keep the updated value in memory, for a future soft-reset. This prevents backtracking of former random sequences. A future soft-reset adds a soft-reset counter and re-hashes it. This provides statistical variation among soft-resets (general purpose seeds follow that same procedure). Disclosure of the updated seed, however, makes future sequences predictable. Hence, a status indication field (using the *.noinit* PUF attribute) can report the PUF status persistently. CSPRNG initialization can follow its own policy to accept or reject seeding after soft-reset.

### 10.4.5 Key Generation

Our key generator follows the approach of the code offset method [182]. Deployments of such a system consist of two phases, namely the enrollment (Figure 10.6a), which has to be executed in a trusted environment, and the reconstruction (Figure 10.6b), which reflects regular device operation.

**Enrollment.** Our key generator provides two enrollment options. (*i*) Helper data is calculated on the device itself. This greatly simplifies a deployment and allows for re-enrollment during deployment time (*e.g.,* via firmware updates). Re-enrollment must be authenticated, though, to prevent invalidation of intact helper data. Self-assessment takes a reference measurement utilizing a low-power power-cycle. A true randomness source is required to generate the random code offset [182] (*cf.* Figure 10.6). We utilize the PUF based secure seed (see Section 10.4.4)

to initialize a crypto-secure SHA256PRNG, which provides unpredictable code offsets of configurable lengths. (*ii*) Helper data is calculated externally, which is convenient for devices with very limited hardware resources. Thereby, a reference SRAM readout is transmitted via UART and an external (trusted) party deals with code offset generation and encoding. In turn, helper data is formatted into a header file that is part of the subsequent compilation of the firmware. This option requires individual compilation for every device to deploy.

For the error correction scheme, we rely on lightweight alternatives, namely, a concatenation of the Golay [127]- and repetition codes, which provide output bit error probabilities of approx. $10^{-5}$ to $10^{-7}$ for common PUF failure rates and lengths [59]. Our modular OS integration allows a seamless replacement of corrections codes in the future.

**Reconstruction.** A device can reconstruct the key after a power-off cycle, utilizing the helper data. After error correction, the key is calculated by a secure hash (SHA256 by default) and stored in a reserved key variable (see Section 10.4.2) to prevent overwriting by subsequent OS startup code. Isolated memory resources are more secure and could hold keys in future, if available on the hardware platform.

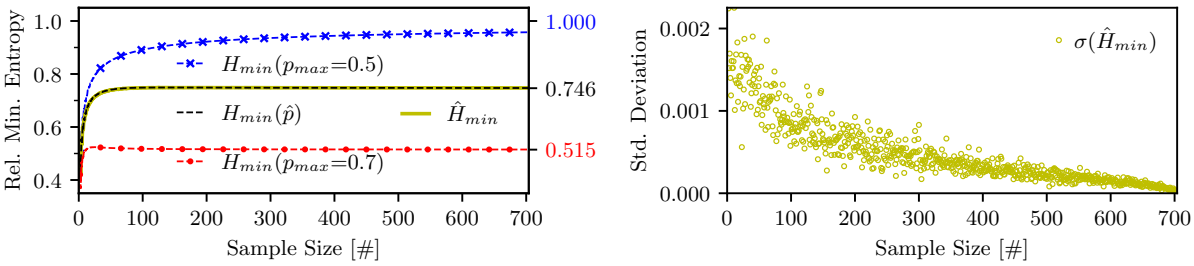
### 10.4.6 Access to PUF Primitives

Random seeds and the secret key are not directly accessible by the user to prevent unauthorized readout, misuse, or tampering. Instead, this vulnerable data is utilized during module initialization, before application code starts in `main`. Seeds are consumed on (CS)PRNG initialization and further processed to obfuscate secret start values, as well as to prepare for the case of a future soft-reset (Section 10.4.4). As a result, application code simply faces a readily usable (CS)PRNG. The secret key can be utilized for the initialization of consuming modules, *e.g.,* as a master key for deriving additional keys that bootstrap security protocols, or to decrypt secured storage. By the end of the module initialization, the PUF derived key is erased, to prevent direct access by the `main` application. Hence, it does not persist through a soft-reset but requires a real power-off cycle to be re-generated. Alternatively, the key can be stored in isolated memory with controlled access in the future.

## 10.5 Evaluation of OS-integrated SRAM PUFs

### 10.5.1 Estimation of the Min. Entropy Convergence

**Bitwise Inter-device Minimal Entropy.** We want to evaluate the unpredictability of uninitialized SRAM between multiple devices using the min. entropy. (*i*) Based on experiment data, we measure the relative frequency $p_{max} = max(p, 1-p)$ for attaining one ($p$) or zero ($1-p$) at the same SRAM bit position of the different devices. Based on a vector of $p_{max}$ values for every bit position, we evaluate the empirical min. entropy for varying sample sizes (*cf.* Equation 10.2) For this, we pick ten sets of devices randomly, and calculate their average




(a) Convergence of the min. entropy estimators for different bias values

(b) Standard deviation of min. entropy measurements

Figure 10.7: Expectation and measurement of the min. entropy for varying max. probabilities ($p_{max}$) and increasing sample sizes.

min. entropy and $p$-values. (*ii*) An estimator theoretically calculates the expected min. entropy or the empirical estimator as a function of the sample size, *i.e.,* the number of nodes, and the maximum probability for logical zero or one.

**Robustness of Estimator.** To assess the validity of our min. entropy measurements, we evaluate its convergence rate. We compare our measurements with a sequence of perfect Bernoulli trials and quantify the convergence for different values of $p_{max}$ (*cf.* Section 10.1.2).

Figure 10.7a presents the results with convergence limits labeled at the right y-axis. For different $p_{max}$ values, the estimated convergence rate varies. Exemplary, a $p_{max}$ of 0.7 decreases the number of samples needed for convergence, but it also decreases the relative min. entropy $H_{min}(p_{max} = 0.7)$ down to $\approx 0.5$. In contrast, the ideal case of $p_{max} = 0.5$ should converge to $H_{min}(p_{max} = 0.5) \approx 1$, which however does not occur within 700 displayed samples. This demonstrates the need for large sample sizes.

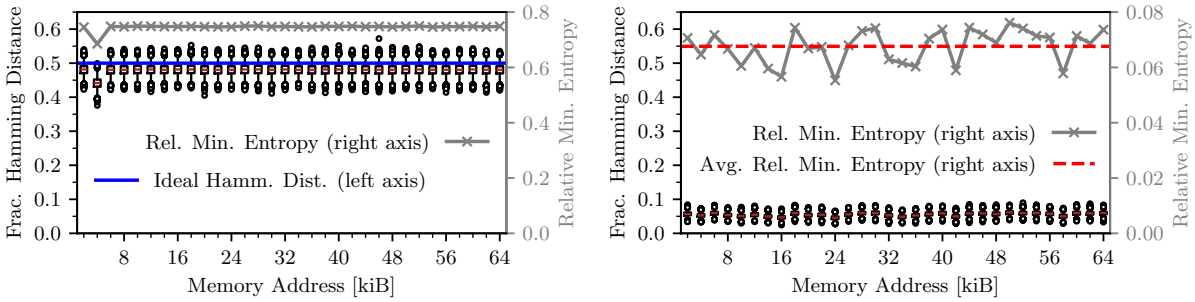
In our measurements, we find a relative frequency of $\hat{p}_1 = 0.596$, which slowly converges to a min. entropy of $\hat{H}_{min} \approx 0.749$ after more than 125 samples. The standard deviation of our measurements $\sigma(\hat{H}_{min})$ yields $2.3 \cdot 10^{-3}$ at max. (Figure 10.7b), and decreases with increasing sample sizes. A comparison of measurement results with our empirical estimator shows almost perfect agreement. We conclude that our measurements with 708 nodes are empirically robust.

### 10.5.2 Blockwise Evaluation of the Uniqueness

**Evaluation between Devices.** We want to quantify the device uniqueness and analyze the fractional hamming distance [165] between devices and blocks, as a preparation to derive unpredictable secrets:

$$HD(r_a, r_b) = |\{r_{a,i} \neq r_{b,i} : 1 \leq i \leq m\}| \cdot \frac{1}{m} \tag{10.7}$$

where $r_{a,i}$ and $r_{b,i}$ denote the bit values of two devices at position $i$ in a block of the length




(a) Between 708 nodes.

(b) Between ≈ 700 reboots on one device.

Figure 10.8: SRAM evaluation. A total of 64 kB SRAM is split and analyzed in blocks of 1024 Bytes. Boxes show fractional hamming distances between all blocks (left y-axis) and lines show min. entropies (right y-axis). IQR: 25th–75th percentile, whiskers: Q1-1.5·IQR and Q3+15·IQR.

$m$=1024 Bytes. Figure 10.8a displays our results for the fractional hamming distance between unique device pairs. Assuming a location-independent occurrence of zeros and ones, the ideal distance is 0.5. Our measurements fluctuate around an average value of 0.48 except for the block at 4 kB (bootloader, *cf.* Section 10.3.3), a slight deviation from the optimum case. Based on these results, we consider memory pattern as unique.

Figure 10.8a additionally presents the blockwise min. entropies (*cf.* Equation 10.1) between all devices as a lower bound of its uniqueness. The min. entropy is commonly used to determine input lengths in crypto-contexts (*e.g.,* key lengths). Our results reveal a min. entropy of ≈ 75 % for each block, which is in agreement with Section 10.5.1 and sufficient to derive unique secrets. As an example, a naive key generator would require 171 Bits of uninitialized memory to create a 128 Bit maximum entropy key.

**Evaluation on a Single Device.** We apply the same methodology to ≈ 700 readouts on the same device to quantify its initial randomness, required to derive distinct seeds. Thereby we utilize a low-power cycle with a sleep delay of one second. Figure 10.8b presents our results for the blockwise hamming distances and min. entropies. The intra-device hamming distances reveal a different picture than the inter-device analysis. Even though a majority of bits remain stable over retries, a small portion adds noise, which leads to intra-device distances of ≈ 0.06 (average). This behavior remains stable among all memory blocks. Bit flips lead to an intra-device min. entropy of 6.8 %±0.51($\sigma$), which supports seed generation. Conversely, a reproducible key generator must eliminate these. To dimension sufficient correction schemes, we also search for the bit error probability in every block and between all measurements, and find the maximum at $p_e$=0.086.

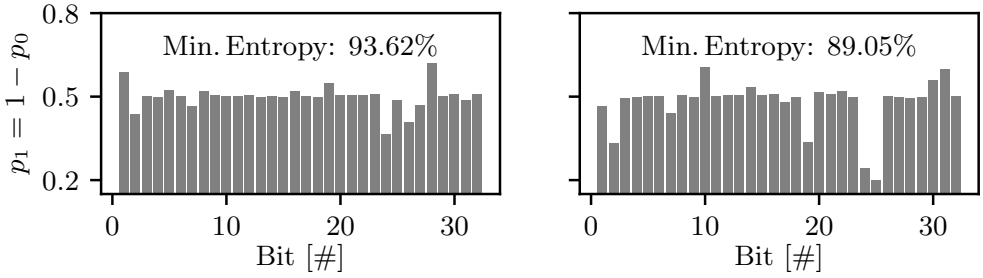



Figure 10.9: Evaluation of general purpose seeds. Index based distribution of bit probabilities throughout 32 Bit integer values and min. entropy per device.

## 10.6 Analysis of Seed and Key Generation

### 10.6.1 Analysis of Random Seeds

We evaluate the quality of seeding and generate two seeds on each startup, (*i*) a secure 256 Bit seed with maximum entropy, (*ii*) a 32 Bit general purpose seed for non security purposes. Our evaluation program triggers periodic power-off cycles of 1 sec. over two days, which results in ≈ 180 k values per device, and 45.1 Mbit secure / 5.7 Mbit general purpose seed bits.

**Secure Seeds.** We calculate the required bits from SRAM based on the intra-device min. entropy of ≈ 7 % as obtained in Section 10.5.2. We account for the entropy loss using the leftover hash lemma [37] ($L = log_2(1/\epsilon)$ with $\epsilon = 2^{-256}$ close to uniform) while targeting at 256 Bit entropy in our final seeds. This requires a minimum of 7314 Bits/914 Bytes of uninitialized memory. We conservatively chose 1024 Bytes. It is worth noting that SRAM portions should be chosen based on a deployment specific initial evaluation of SRAM properties. All seed values are unique and uniformly distributed due to the properties of the SHA256 hash.

**General Purpose Seeds.** A min. entropy of 7 % requires a minimum of 457 Bits/57 Bytes of SRAM to provide 32 Bit of seed entropy. Conservatively, we choose 128 Bytes with well aligned values in return. Figure 10.9 presents the probabilities of $p$ for every bit in the 32 Bit seed, from two sample devices. They roughly follow a normal distribution and provide 89–95 % min. entropies, which we consider sufficient for non-security purposes.

### 10.6.2 Analysis of the Fuzzy Extractor for Key Generation

Figure 10.10 visualizes the fuzzy extractor properties for varying configurations (*cf.* Section 10.4.5). Similar to the seed evaluation, every configuration produces ≈ 180 k values. We vary the code offset from 9 to 24 Bytes on the y-axis, and the number of repetitions by the repetition error-correction code between 1–13 on the x-axis. A repetition of 1 reflects a single occurrence of the code word. The Golay code is active in all cases. Lower right triangles

in Figure 10.10 (blue) encode the length of required SRAM bits. The same length is required for helper data on non-volatile memory.

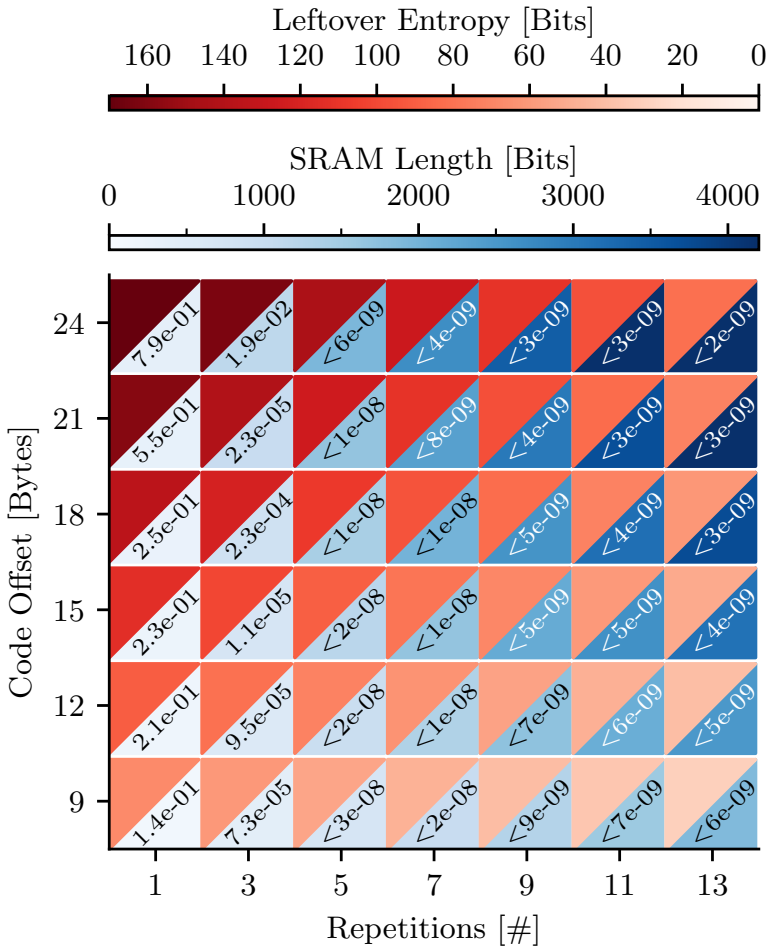



Figure 10.10: SRAM length, remaining entropy, and measured reconstruction failure rate for different configurations of the fuzzy extractor.

**Remaining Key Entropy.** A naïve estimation of the entropy of a key output would multiply the SRAM length by the inter-device min. entropy to determine its cryptographic strength. For example, a code offset of 9 Bytes with repetition 1 leads to an SRAM length of 18 Bytes/144 Bits; multiplied with a min. entropy of $\approx 0.75$ would then yield 108 Bits of entropy in the SRAM used for key derivation. For biased SRAM, however, publicly available helper data leak information about the generated key [206, 86, 250], which is due to the concatenation of the two error correction codes as part of the fuzzy extractor. This leakage further reduces the remaining entropy in the key and requires additional random code offset- and SRAM bits to compensate. Maes *et al.* [250] derived methods for calculating the leakage and the remaining entropy as a function of bias, which reflects the average-case resistance against brute force attacks [257].

We determine the remaining entropy for varying fuzzy extractor configurations and for our measured SRAM bias of $\hat{p}_1 = 0.596$ (*cf.* Section 10.5.1). Figure 10.10 visualizes the results for various configurations of the fuzzy extractor. The upper left triangles (red) reflect the remaining key entropy after fuzzy extraction. Increasing code offsets increase the required SRAM length (*i.e.,* initial entropy) and the remaining entropy in the extracted keys. Increasing repetitions unsurprisingly increase the required input lengths too, whereas the remaining entropy shows a reversed trend and increases with fewer repetitions. Code offsets of 24 Bytes expose remaining entropies from 182 Bits (1 repetition) down to 82 Bits (13 repetitions). A random code offset of 24 Bytes with 5 repetitions provides 144 Bits of remaining entropy and meets the recommended security strength [39] of 128 Bits key entropy.

**Reliability.** Figure 10.10 also presents the empirical reconstruction failure rate, which is introduced by bit errors between SRAM readouts that cannot be corrected by the fuzzy extractor. Increasing code offsets increases the error rate (notable in Figure 10.10 following repetitions 1 and 3 for bottom to top). Following repetitions fewer than five, all fuzzy extractor configurations reveal a notable failure probability, which contradicts the common key reconstruction error rate of $10^{-6}$ [138, 374, 250, 172]. Five or more Repetitions expose errors smaller than $3 \cdot 10^{-8}$. In our measurements, no uncorrected bit error occurred in reconstructed outputs and the error values represent the multiplicative inverse of all successfully reconstructed bits.

**Discussion.** Increasing the SRAM length is undesirable since memory is sparse on very constrained IoT devices. Repetitions should remain few to avoid entropy loss. Conversely, multiple repetitions are required to provide an acceptable reconstruction rate, in particular for deployments of large SRAM noise level [332]. A code offset of 24 Bytes and five repetitions preserves sufficient key entropy on our *M3* nodes at a failure rate that meets the requirements for a PUF design. Other fuzzy extractor configurations either sacrifice reliability by an intolerable reconstruction failure rate at the required level of security, or they sacrifice the remaining key entropy. Our overall balanced strategy provides highly unique and reliable device identities at an acceptable security level. Pre-processing of the SRAM pattern as proposed in [404, 312, 242] can further reduce the required SRAM length and increase the remaining key entropy from biased SRAM PUFs, which promises to improve the performance at the same or better security strength.

### 10.6.3 Resource Overhead

**Processing Time.** We measure processing times on *M3* nodes and compare the PUF performance with two different off-the-shelf IoT platforms: *ESP32* and *HiFive* (see Section 10.2).

First, we analyze the startup latency of two RIOT applications executed on the *M3* node, without the PUF module. (*i*) `Hello world` is a minimal single-threaded application and introduces a startup latency of 1.1 ms. (*ii*) `gnrc_networking` is the standard IPv6 networking application which initializes many modules in 8 threads prior to execution of application code.

Table 10.1: Additional operating system startup latencies introduced by soft-reset detection and generation of two seeds.

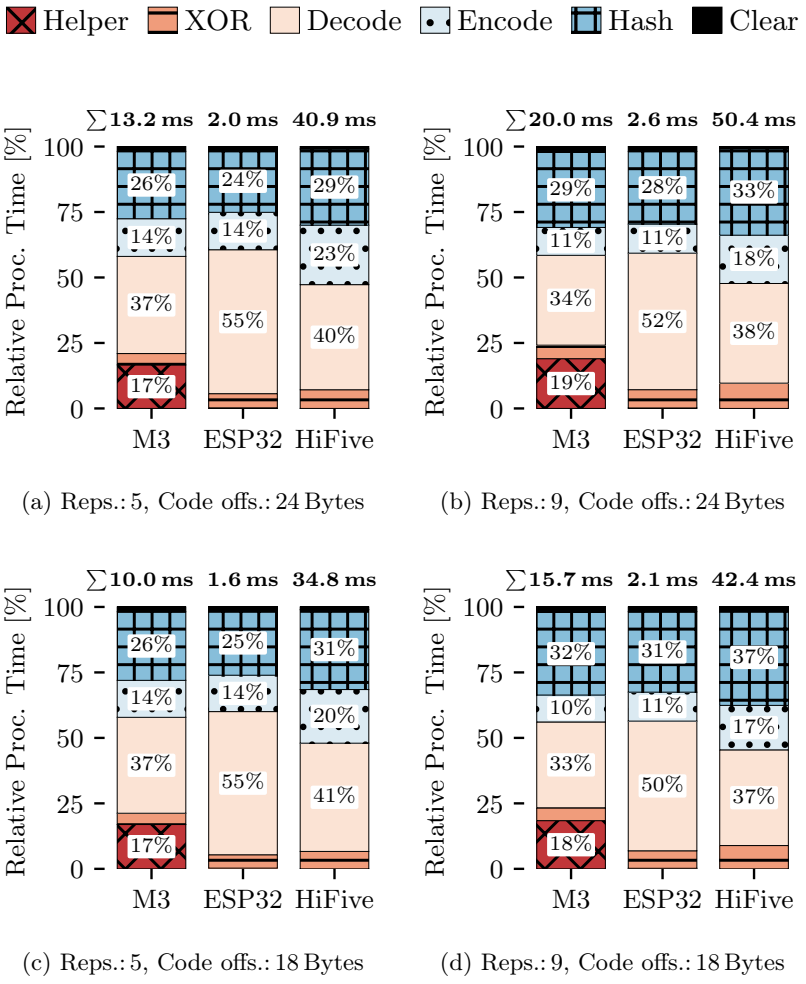
| Platform | Soft-reset detection [ms] | Common routines [ms] | Seed generation [ms] | |
|---|---|---|---|---|
| | | | simple | secure |
| *M3* | $7.65 \cdot 10^{-3}$ | $3.00 \cdot 10^{-3}$ | 0.13 | 13.65 |
| *ESP32* | $22.59 \cdot 10^{-3}$ | $0.47 \cdot 10^{-3}$ | 0.02 | 1.37 |
| *HiFive* | $494.91 \cdot 10^{-3}$ | $1.50 \cdot 10^{-3}$ | 0.08 | 27.03 |

This requires 10.8 ms for startup. The latter case excludes seed generation. Here, we utilize a static CSPRNG seed, since microcontroller lacks an entropy source (without the PUF).

Table 10.1 presents the processing overhead of (*i*) soft-reset detection, (*ii*) common routines, and (*iii*) both seed generators. Soft-reset detection is mandatory with our PUF module and adds a small overhead of $< 8\,\mu$s on the *M3* node. *ESP32* adds $\approx 23\,\mu$s and *HiFive* surprisingly requires $\approx 65$ times longer than *M3*. This is an effect of PUF operation prior to system clock initialization. Common processing adds a negligible overhead on all platforms. General purpose seed generation ($\approx$ 0.02–0.13 ms) is lean compared to secure seed generation (up to 14 ms on *M3*) which is comparable to `gnrc_networking`, though, seed generation from real entropy sources is slow in general [199]. Secure seeds take $\pm$ 12 ms on *ESP32* and *HiFive*. Flash memory access during SHA256 computation is slower on *HiFive* due to a serial interface. In agreement with previous measurements, processing times do not directly reflect CPU frequency. Initializing clocks prior to PUF execution can improve performance in the future, but requires rearrangement of the OS startup routine. Exemplary, we rearrange the startup code for the *M3* platform and find a speedup of almost 7 times, though, system clock speed increased by a factor of 9, comparing the hardware default state (8 MHz) and the RIOT configuration (72 MHz).

Next, we look at the processing overhead of the fuzzy extractor and focus on reconstruction since enrollments happen rarely. We present four relevant configurations for key construction in Figure 10.11. 'Helper' contains readout of the helper data from flash. 'XOR' contains the overhead from bitwise xor operation at the input and output of the fuzzy extractor (Figure 10.6a). 'Decode' includes overhead of the concatenated Golay- and repetition decoder, and 'Encode' includes renewed encoding of the corrected code offset. 'Hash' calculates a digest over the reconstructed PUF measurement. Finally, 'Clear' contains the overhead of re-setting vulnerable data structures after usage.

The absolute latency (numbers above bars) depends on the SRAM length and requires 10–20 ms on *M3* in all presented cases. The order of magnitude compares to `gnrc_networking` and the secure seed generator. Reconstruction and seed generation add to the existing startup latency, though. Other platforms reflect results from Table 10.1 and take 1.6–2.6 ms (*ESP32*)




(a) Reps.: 5, Code offs.: 24 Bytes

(b) Reps.: 9, Code offs.: 24 Bytes

(c) Reps.: 5, Code offs.: 18 Bytes

(d) Reps.: 9, Code offs.: 18 Bytes

Figure 10.11: Additional OS startup latency introduced by PUF reconstruction for four configurations of the fuzzy extractor on different boards.

and 35–50 ms (*HiFive*) respectively. Readout of the helper data is only notable on the *M3* (≈ 17 %) due to its slow NOR flash. The relative processing time for fuzzy extraction increases almost linearly with longer code offsets (Figure 10.11 bottom to top). Increasing the number of repetitions (Figure 10.11 left to right) also increases the relative hashing time for a reduction in decoding. Longer inputs affect the cryptographic hash efforts moderately more than the simple decoder.

In summary, the collection of PUF features moderately delays the startup routines of our sample applications. This motivates our modular design, which allows for selective configuration of PUF features. Furthermore, a positive soft-reset detection skips parts of the PUF execution. The order of tens of milliseconds is still small compared to the required SRAM power-off time (1 second has proven suitable for different platforms) to generate a fresh memory pattern. In

Table 10.2: Current- and energy consumption of the PUF for four configurations of the fuzzy extractor, and a comparative use case—measured on an *M3* node.

| **Reps. [#]** | **5** | **9** | **5** | **9** | *Comparative* |
|---|---|---|---|---|---|
| **Code offs. [B]** | **24** | **24** | **18** | **18** | *case* |
| **Avg. Current [mA]** | 8.26 | 8.19 | 8.28 | 8.07 | *44.72* |
| **Energy [$\mu$J]** | 757.49 | 941.88 | 681.06 | 807.21 | *439.52* |

practice, most IoT applications only awake a few times per hour or day, which obviates the latency overhead.

**Energy Consumption.** Table 10.2 presents the average current flow and the energy consumption of PUF execution on the *M3* node, including the generation of two seeds following four configurations of our fuzzy extractor, in agreement with Figure 10.11. To compare these results, we have added a *Comparative case* which acts as a simple alternative for seed- and key inclusion without the PUF. Thereby, we create two seeds by requesting noise from the external radio module (without further conditioning) on the *M3* board and assume a pre-provisioned key to persist in external flash memory, which we import. The quality of randomness, unpredictability, and uniqueness, in this case does not compare to the PUF. PUF execution prior to systems initialization drains a small current of less than 8.3 mA on average, whereas the energy consumption ranges from 680–942 $\mu$J, depending on the input length. This is in line with the processing time. Our *Comparative case* drains more than five times higher current compared to the PUF, for two reasons. First, this case additionally requires the radio module to be powered. Second, it is operated after systems initialization. This speeds up the execution time ($\approx$3 ms), however, a higher clock speed further increases the MCU current, leading to a total energy consumption of $\approx 440\,\mu$J. In summary, the energy demands of the PUF are on the same order of magnitude compared to a simplified use case, and increase the consumption by a factor between 1.5 and 2. Therefore, our PUF contributes conditioned seeds and uniform keys across devices, at little current drain, without depending on additional external hardware modules on the boards.

Table 10.3: Threat overview of the SRAM PUF integration.

| No. | Threat description | Asset (§10.7.1) | Adversary (§10.7.2) | Surface (§10.7.3) | STRIDE (§10.7.4) | Mitigation |
|---|---|---|---|---|---|---|
| T0 | Readout public helper data. | A5 | Hardware | S2 | I | • Conserv. entropy estim. during enrollment. |
| T1 | Read/write data. | A1–5 | Hardware | S2 | STRIDE | • Enable debug port lock.<br>• Use one-time program. memory / write protect.<br>• Cut input/output connections.<br>• Deploy device with tamper protect. enclosure. |
| T2 | Manipulate operational conditions. | A1 | Hardware | S3 | TID | • Additional entropy sources for seed generation.<br>• Sensors to monitor environ. conditions [18]. |
| T3 | (Crypto-)analysis of network traffic. | A1–3 | Software | S1 | I | • Conserv. entropy estim. during enrollment.<br>• Short error correction codes. |
| T4 | Readout public/secret data. | A5 | Software | S1 | I | • Clear memory after usage.<br>• Separate mem. for non-/secure seeds and key. |
| T5 | Overwrite control data. | A4–5 | Software | S1 | TRID | • Enable hardware assisted soft-reset detection. |
| T6 | Control operational conditions. | A1-3 | Software (&Hardware) | S1 | TID | • Enable hardware assisted voltage detection. |

## 10.7 Security Analysis

PUFs need to maintain unpredictability and unclonability. A secret is embedded in the chip, hence, no seed or key is stored during device sleep, the prevalent state of a battery-driven IoT device. Secrets only persist during a short time after system startup, reducing the attack vector to a limited time. Practical attacks, however, can still exploit a number of vectors. We identify (*i*) assets, (*ii*) attackers, and (*iii*) attack surfaces of our PUF module, and present (*iv*) threats. Risks arise from the combination of specific hardware capabilities, the deployment consideration, application requirements, and related attacker assumptions. Hence, we are aiming to provide an overview of the prevalent risks, together with a series of mitigations.

### 10.7.1 Assets

The most vulnerable resources of the SRAM PUF are the uninitialized memory pattern (**A1**), the output of the PUF, namely secure seeds (**A2**), and the key (**A3**). These assets must preserve confidentiality and integrity. In our implementation, the memory marker (**A4**) (*e.g.,* for soft-reset detection, see Section 10.4) persist after OS startup and is vulnerable because it controls the next reset behavior, *i.e.,* can instruct to skip or execute the PUF on a future reset. Hence, this data must preserve integrity. Non-volatile memory (**A5**) stores helper data that is required for key reconstruction. Although helper data is considered public, it is still susceptible. It must preserve integrity and availability to reconstruct the PUF correctly. Authenticity is also desired, but conventionally very challenging to achieve.

### 10.7.2 Adversaries

We distinguish two types of adversaries. First, **software** attackers that try to compromise, manipulate, or analyze the system under attack without hardware access. This includes crypto-analysis and the application of learning algorithms. Software attackers exploit software back-doors, weak implementations, or software bugs to reveal secret information, or disturb code execution. Considering networked nodes in the IoT, attackers can be in wireless reach or connected remotely. Second, **hardware** attackers that have direct physical device access. We distinguish two types of hardware attackers: *Non-invasive* attackers try to interface the device during sleep or operation. They utilize interfaces such as system peripherals, or try to manipulate the device operation conditions. *Invasive* hardware attackers have deep knowledge and access to advanced techniques to gather or manipulate information on the silicon level. We exclude *invasive* attacks from the remainder of this section because they are (*i*) rare due to high financial and knowledge requirements and (*ii*) very specific to chip constructions, and so are mitigations, which contradicts our goal to improve the security of cheap, heterogeneous, and possibly already deployed devices.

### 10.7.3 Surfaces

We categorize the attack surfaces into three groups. (*i*) The communication interface (**S1**), *e.g.,* the low-power radio can act as an entry point to inject malicious inputs, or be used for (crypto-) analysis of protocols that make use of random numbers derived by the PUF seed, or the key derived by the fuzzy extractor. This interface also acts as entry point for software updates (future work). (*ii*) I/Os provide an interface to the MCU (**S2**). Peripherals such as UART, SPI, or GPIO can revel system internals through logging output, and open an attack vector for interaction with the system. More crucial, debugging interfaces such as JTAG open a direct interface to the chip memory. (*iii*) The physical presence of a device (**S3**) provides a surface to operational conditions (*e.g.,* temperature, magnetic field) and the power supply.

### 10.7.4 Threats & Mitigations

We classify threats using STRIDE [207] which defines six categories of security threats: Spoofing identity (**S**), Tampering with data (**T**), Repudiation (**R**), Information disclosure (**I**), Denial of service (**D**), and Elevation of privilege (**E**). Table 10.3 summarizes our results and presents mitigations for hardware (**T0**–**T2**) and software (**T3**–**T6**) adversaries.

**T0.** An attacker manages to read non-volatile memory, by (physically) connecting to the flash memory. Without the PUF, persistent keys would be stored as plain text, directly disclosing the secret. PUFs provide additional security by storing only the public helper data in flash. This attack, however, may disclose information in cases of high bias. Hence, helper data readout should still be impractical.

**T1.** An attacker manages to read/write data such as the uninitialized SRAM pattern, seeds, or keys. Debug interfaces can directly interact with the processor. Adversaries that manage to connect to the debug lines and initiate a debug session, can halt the CPU during startup to read out memory. If the PUF primitive is used for authentication, this enables spoofing and elevation of privileges without repudiation. Tampering can invalidate operation leading to denial of service which, however, is simple to achieve with physical device access. It is noteworthy that PUFs do not introduce additional threats compared to pure software solutions.

**T2.** An attacker manages to tamper by manipulating environmental operation conditions of the device. Common examples vary the temperature or control the power supply, *e.g.,* the power-off time, operation voltage, or startup slope. This affects random physical processes, including but not limited to SRAM startup state. A reduction of entropy disqualifies seeds and discloses information, especially in combination with T0. False key reconstruction can lead to denial of service. Without the PUF, applications require alternative sources for seed generation, or sometimes use TRNGs permanently which are similarly affected by the environment. PUFs thus act as an additional source of entropy to increase seed security.

**T3.** An attacker monitors (secured) network traffic that utilizes random numbers or keys. This attack might be complemented by owning and analyzing the SRAM on a device of the same type,

exploiting bias to predict the initial SRAM pattern. Crypto-analysis of the output of known algorithms can disclose information of secrets derived from insufficient entropy. Combined with T0, learning attacks become a risk [357] in these cases. Without the PUF, however, random numbers are unavailable on platforms without a TRNG which fully prevents security. Keys are sometimes shipped by the vendor and reutilized across devices, leading to zero entropy on large quantities of nodes [317, 364]. PUFs enable security contributing a uniformly random seed and a unique key that is derived from individual device variations.

**T4.** An attacker manages to read data structures through software backdoors, which challenges privacy regardless of PUFs (*e.g.,* compromise of keys in working memory). At the time that the network interface is up and running, SRAM is not uninitialized anymore, and vulnerable seeds should be cleared. A state compromise of a non-forward secure PRNG, however, potentially allows backtracking of the initial SRAM pattern and discloses information. Without the PUF, initial secrets are likely stored persistently in plain text. PUFs reduce this attack surface to helper data disclosure (see T0).

**T5.** An attacker manages to overwrite vulnerable data structures (*e.g.,* forcing buffer overflow). Tampering with the soft-reset memory marker (Section 10.4.3) can trigger a false negative detection on next reset, which leads to zero entropy seeding and defect key reconstruction. Similarly, tampering helper data is a risk. This causes information disclosure and enables denial of service without repudiation. Interfering with code execution threatens code execution regardless of PUFs, though.

**T6.** Combines T2 and T5. An attacker manages to tamper with the voltage supply through **software** interfaces–likely present in low-power OSs for undervolting. Dynamic adjustments during program execution that do not affect startup conditions after reset (sleep) are uncritical, since the PUF is processed before operation. Adjustments that persist after reset, however, are crucial. A lower voltage causes the reduction of SRAM entropy which disqualifies seeds, leading to information disclosure. Alternative random sources might be similarly affected by this attack (see T2).

**Threat discussion.** PUFs provide non-uniform keys across devices, which means that each device has to be attacked separately, rather than attacking one and owning all devices. The success of a hardware attacker depends on (*i*) the device accessibility and (*ii*) the additional security features of the chip. Hardware attacks are typically small-scale, which contradicts the large-scale characteristics of common IoT deployments. A *non-invasive* hardware attack requires high efforts for a single device, whereas many threats can be mitigated by standard hardware features. High security applications, however, should design specific hardware-security features and consider device enclosures.

Software attacks are more likely in the IoT since devices become accessible remotely through the network. Thereby, the attack surface reduces notably, compared to hardware attacks. Presuming an adequate enrollment, the prevalent software-based threat is given by information

disclosure and tampering through a software backdoor (T4–T6). These threats, however, do not assault the PUF in particular, but generally impede operation of this constrained device class. Hence, the PUF adds a layer of security in practice. To reduce this attack surface, vendors include trusted execution environments (*e.g.,* ARM TrustZone, RISC-V PMP) on modern IoT platforms, that allow for code isolation and privileged memory access. Privileged PUF operations can improve security by separating user facing, networking, or driver code from secure operations. Conversely, secure processing environments require a root of trust, which can be assisted with a PUF. Hence, both features could complement each other in the future.

## 10.8 Conclusions

This chapter started from the observation that many commodity IoT devices provide little to no hardware security features, sometimes not even a source of randomness. We presented the first comprehensive PUF integration into an IoT operating system to fill this gap and broadly enhance embedded security. Our PUF proposal uses uninitialized SRAM, which is available on common IoT platforms, and is portable due to an integration below the hardware abstraction layer of the open-source operating system.

We evaluated SRAM PUF on typical class 2 devices in an open testbed using 708 nodes. This is, to the best of our knowledge, the first empirical PUF study with several hundreds constrained IoT nodes, albeit prior work [397] proved the need for large sample sizes for the subtle analysis of SRAM bias. Our analysis revealed four key insights.

(*i*) An inter-device distance of $\approx 48\,\%$ between node pairs shows high uniqueness, which enables the generation of unpredictable keys. Still, the physical SRAM layout introduces inter-device bias, which becomes visible when analyzing high numbers of nodes. This reduces the inter-device min. entropy to $\approx 75\,\%$, and thereby the number of unpredictable bits per node. Key generation relies on public helper data, which may reveal information about the SRAM pattern in the case of bias. Our analysis of the entropy leakage identified a fuzzy extractor configuration that results in 144 Bits of remaining key entropy at a failure rate of $6 \cdot 10^{-9}$. (*ii*) An intra-device min. entropy of $\approx 7\,\%$ allows for secure seed generation on startup. (*iii*) The uninitialized SRAM properties of real-word aged, heavily utilized testbed nodes are still sufficient to achieve (*i*) and (*ii*). (*iv*) A configurable OS integration can seamlessly provide PUF services to the IoT at moderate start-up overhead while shielding soft resets.

We could also show that a number of hardware-based *non-invasive* attacks against SRAM PUFs heavily depend on the availability of platform features such as device pinouts or debug port locks. The availability of PUFs upgrades the security of commercial off-the-shelf devices without cryptographic hardware and strengthens the resistance against the more dangerous software attacks from remote parties throughout the Internet. Contributing non-uniform keys

across devices, our PUF integration reduces the efficacy of these attacks, since each node needs to be attacked individually, rather than attacking one and owning all.

**Acknowledgments.** We would like to thank Nils Wisiol for his careful feedback, which has significantly helped to improve the chapter. This work was supported in part by the German Federal Ministry for Education and Research (BMBF) within the project *PIVOT: Privacy-Integrated design and Validation in the constrained IoT.*

**Availability of software and reproducibility.** We support reproducible research ([4, 327]) and utilize open source software and open testbed platforms. All of our work is publicly released. The code of the software components, pre-compiled binary images, the implementation of the estimator, documentation, data sets and related tools are available on GitHub at `https://github.com/inetrg/IEEE-TDSC-PUF23`.

# Chapter 11

# Conclusions and Outlook

Unreliable wireless transmissions, and the need for protection against threats of the Internet, impose severe challenges to resource-constrained IoT networks. We discussed detailed conclusions and aspects of future work in the respective chapters. In the remainder of this chapter, we summarize the overall conclusions and survey the directions for future research.

Information-centric networking is beneficial for the IoT. Instead of maintaining stable communication channels like in IP-based protocols, hop-wise data transfer and content caching increase reliability in low-power lossy networks, reduce bandwidth demands, and improve the resource utilization on constrained nodes. The lack of a name to MAC mapping, however, leads to an energy excess by broadcast forwarding, which conflicts with limited device resources. This motivated our work on the link layer convergence in ICN. We designed ICN over DSME, using LoRa as a transmission technology. Our system design enables robust communication for long-range radios, and a decentralized (information-centric) network architecture with caches facilitates edge deployments that preserve limited energy resources on battery-driven nodes. Inter-connecting these networks with the Internet, however, is challenging due to a different understanding of timescales. Sporadic IoT data generation, and extremely long producer delays challenge the pull-based ICN scheme and quickly lead to data polling, demanding for delay-tolerant ICN-extensions which we provided in this thesis.

Our crypto-integration into a commodity IoT operating system facilitates the deployment of protocols with full security features enabled, using hardware accelerated cryptography instead of software. But there is a caveat: Random generating hardware, *e.g.,* peripheral TRNGs, sometimes exhibit a high energy consumption and poor statistical properties. Hence, our randomness OS-subsystem prefers software (CS)PRNGs, perhaps complemented with accelerated crypto-operations, instead of directly mapping to random generating hardware peripherals below the user interface. An embedding in the ecosystem of the operating system enables entropy gathering from different hardware resources through the hardware abstraction layer, to generate reliable (CS)PRNG seed values. Many low-end IoT devices lack hardware security features. Our SRAM PUF OS-integration provides a zero-cost approach that bootstraps the cryptographic subsystem, contributing seed and key material from hardware variations. We hope this will ease deployment efforts and contribute to a more secure IoT in the future.

**Potentials of ICN for Constrained IoT Networks**. Based on experiments from real deployments of host-centric protocols (CoAP, MQTT-SN), and information-centric networking (NDN) in a testbed, we gained insights on their performance in constrained low-power networks. Our competitive benchmarks revealed that push-based protocols (*e.g.,* CoAP Observe) operate fastest in single-hop scenarios. In challenged multi-hop scenarios, the pull-based NDN flow with hop-wise data transfer and caching outperforms IP-based protocols, which initiate end-to-end communication channels without intermediate caching (*e.g.,* CoAP GET). The data-centric approach increases reliability by up to 100 % compared to host-centric protocols while reducing corrective actions (retransmissions), and reduces latency and link stress. Producer-initiated data transfer, however, is not available in plain NDN and requires publish-subscribe extensions (HoPP), that standard protocols natively inherit.

**MAC Address Mapping in ICN**. Broadcast forwarding on the wireless medium simplifies content distribution in NDN, but includes two major drawbacks. (*i*) Broadcast excludes error handling mechanisms of the link layer, which increases the rate of unsatisfied Interest requests up to 12 %, compared to unicast with ARQ. (*ii*) Nodes in wireless reach need to process all packets, which conflicts with resource constraints, *i.e.,* broadcast forwarding increases the active CPU time per node by two orders of magnitude, compared to unicast. Mapping names to the unicast address, however, requires additional logic to learn and maintain routes. The missing link layer convergence in NDN motivated our subsequent work on a new MAC layer for the information-centric IoT.

**Decentralized MAC and Network Layer for LoRa**. Our long-range wireless communication system puts the 802.15.4 DSME MAC layer in place for reliable and bidirectional LoRa communication, and combines the system with information-centric principles. Instead of relying on a centralized network server like in LoRaWAN, the ICN-based system design reduces dependencies on centralized components and exploits ICN features (selective forwarding, Interest aggregation) that reduce wireless link utilization. Based on simulation results, we explored different mappings of ICN message semantics onto the multifaceted MAC layer and find that our system provides horizontal scalability and robust media access for LoRa radios. An extended caching gateway facilitates low-power operation and acts as custodian for sleepy nodes.

**Delay-tolerant Networking with ICN**. Long and vastly differing producer delays challenge information-centric data retrieval on the end-to-end path, since high-speed forwarders on the Internet operate on a different timescale, leading to premature state expiration and polling. Two recently proposed ICN protocol extensions were adopted, namely RICE and reflexive forwarding, to deal with varying and dramatically higher round trip times. Our practical implementation of LoRa-ICN enabled comparative experiments between basic NDN-style communication, and the delay-tolerant alternatives, serving slow edge networks. Executed on off-the-shelf IoT nodes and an emulated Internet, our results revealed that both extensions for Internet consumer-

initiated and producer-initiated communication exhibit almost 100 % data reliability, targeted completion time for sporadic data generation, and lifetimes of over one year for battery powered LoRa nodes, while maintaining core ICN features.

**Analysis and Integration of Cryptographic Backends**. A novel OS-level crypto-subsystem enabled our measurement study for widely used cryptographic primitives across various off-the-shelf IoT platforms. The resource analyses gained two main insights. (*i*) Crypto-hardware outperforms software by more than 100 %, which is crucial for nodal lifetime. (*ii*) External crypto-processors operate slowly on symmetric crypto-operations, but provide write-only key storages, which demands for an identifier based key management. These results are a first step to prevent performance pitfalls in the future. Heterogeneous crypto-backends require different levels of hardware support to utilize features most efficiently, and a configurable environment that models these features at compile time, reducing the memory footprint. Our subsystem isolates this complexity from the developer and contributes to code usability, and portability across devices.

**Random Number Generation in the Low-end IoT**. Random number generation remains an active research topic due to the increasing amount of IoT data, the development and standardization of new protocols that secure network traffic, and the advent of machine learning at the Internet edge that increases the need for random numbers on IoT devices. The landscape of random generating hardware and software that produce statistical randomness or entropy challenges a system design that integrates these components uniformly. Our comparison of state-of-the-art open source OSs shows that randomness building blocks are poorly grounded in the ecosystem and often rely on some external library, which impedes a full exploitation of hardware features (*e.g.,* for entropy gathering) and sometimes ignores device constraints. Based on a comparative evaluation of the statistics, security, and performance on popular IoT platforms, we derived recommendations and requirements that guide the design of systems that operate constrained embedded devices.

**Seed- and Key Generation with Physical Unclonable Functions**. PUFs enable security on low-end platforms that lack hardware-based security features. The quality of seeds and keys derived by a PUF are subject to bias and aging, though. Analyzing these effects requires a large sample size, which we contributed in this thesis. Our SRAM evaluation on 708 nodes showed a static bias at certain bit positions, and stress marks by past experimentation on the nodes from a testbed. An intra-device analysis still revealed sufficient random noise between power-cycles on one device to generate random seeds with 256 Bits of entropy. An inter-device analysis revealed sufficient uniqueness to generate unpredictable keys with 128 Bit of security. Our practical threat model assessed the security of SRAM PUFs. The most dangerous attacks in the IoT are remote attacks which can be performed at a large scale with low overhead, and without physical device access. Here, our SRAM PUF advances the security of devices that lack security hardware and fall back to pure software solutions otherwise.

## Directions for Future Work

Information-centric networking has promising features for the IoT. More large-scale deployment studies are necessary, however, to gain insights on how resource-constrained ICN edge networks, interconnected via multi-tenant gateways, perform at a global scale, and to identify technological gaps in our initial system design. A first step in that direction is the device bootstrapping and trust establishment with a custodial gateway at the edge.

Progressing on the convergence between the ICN network layer and other wireless technologies (Sigfox, NB-IoT) as well as the alignment with recent IETF/IRTF standards, *e.g.,* header compression in wide area networks (SCHC), or ICN over personal area networks (ICNLoWPAN), can increase technological versatility and gain insights about inter-domain performance and scalability of the data-centric approach in the IoT.

Novel protocol standards such as ACE-OAuth, or upcoming standards such as EDHOC and LAKE may introduce notable security overheads and will make heavy use of crypto-operations. The advent of quantum-resistant cryptography will transition cryptographic algorithms used in existing protocols, and may increase the required bandwidth, latency, and energy in the future. The effect of heavy crypto-utilization on the performance of these novel protocols should be investigated to assess the impact on the lifetime of securely networked IoT nodes.

PUFs increase the security of low-cost devices, but are subject to aging. Quantifying aging effects of real-world node operation requires controlled long term experiments that go beyond artificial aging, ideally at large scale to gain statistical significance. This gives insights about the correlation between node utilization, *i.e.,* operated by realistic IoT firmware, and its impact on hardware degradation over the time. As a first step, the effectiveness of existing countermeasures against (artificial) aging can be examined, based on naturally aged nodes. As a second step, this may enable the design of novel runtime countermeasures that comply with resource-constraints, that should mitigate entropy loss on degrading nodes in the future.

New hardware architectures will become increasingly available on modern IoT platforms and should extend our crypto-integration, *e.g.,* Ascon is a novel standard[1] for lightweight cryptography and may replace the common AES cipher suite in constrained networks. Quantum-resistant cryptography may increase crypto-hardware utilization in the future due to an increasing algorithm complexity that conflicts with current IoT device resources. NIST has selected the first four quantum-resistant algorithmic standards[2], and more could follow in the future. Heterogeneous accelerators are likely to appear, which may require additional hardware-software co-designs for configurability. Once available, analyzing the upcoming hardware landscape will act as a starting point to identify suitable abstraction levels for an OS-integration.

[1] `https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon`

[2] `https://nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms`

# List of Figures

# List of Tables

# Bibliography


[1] D. E. 3rd, J. Schiller, and S. Crocker. Randomness Requirements for Security. RFC 4086, IETF, June 2005. URL `https://doi.org/10.17487/RFC4086`.

[2] A. Abrardo and A. Pozzebon. A Multi-Hop LoRa Linear Sensor Network for the Monitoring of Underground Environments: The Case of the Medieval Aqueducts in Siena, Italy. *Sensors*, 19(2):402, 2019.

[3] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the Usability of Cryptographic APIs. In *Proc. of the IEEE Symposium on Security and Privacy (SP '17)*, pages 154–171, Los Alamitos, CA, USA, 2017. IEEE Computer Society.

[4] ACM. Result and Artifact Review and Badging. `http://acm.org/publications/policies/artifact-review-badging`, Jan., 2017.

[5] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, Piscataway, NJ, USA, Dec 2015. IEEE Press. doi: 10.1109/WF-IoT.2015.7389098A. URL `https://doi.org/10.1109/WF-IoT.2015.7389098`.

[6] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Li, S. Mastorakis, Y. Huang, J. P. Abraham, E. Newberry, S. DiBenedetto, C. Fan, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, M. Chowdhury, and L. Wang. NFD Developer's Guide. Technical Report NDN-0021, NDN, August 2021. URL `https://named-data.net/publications/techreports/ndn-0021-11-nfd-guide/`.

[7] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking. *IEEE Communications Magazine*, 50(7):26–36, July 2012.

[8] B. Ahlgren, A. Lindgren, and Y. Wu. Demo: Experimental Feasibility Study of CCN-lite on Contiki Motes for IoT Data Streams. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 221–222, New York, NY, USA, 2016. ACM.

[9] M. Al-Zubaidie, Z. Zhang, and J. Zhang. Efficient and Secure ECDSA Algorithm and its Applications: A Survey. *Int. Journal of Communication Networks and Information Security (IJCNIS'19)*, 11(1), 2019.

[10] J. Alamos, P. Kietzmann, T. C. Schmidt, and M. Wählisch. DSME-LoRa – A Flexible MAC for LoRa. In *Proc. of 29th IEEE International Conference on Network Protocols (ICNP 2021), Poster Session*, Piscataway, NJ, USA, November 2021. IEEE. URL `https://doi.org/10.1109/ICNP52444.2021.9651945`.

[11] J. Alamos, P. Kietzmann, T. C. Schmidt, and M. Wählisch. DSME-LoRa: Seamless Long Range Communication Between Arbitrary Nodes in the Constrained IoT. *Transactions on Sensor Networks (TOSN)*, 18(4):1–43, November 2022. URL `https://dl.acm.org/doi/10.1145/3552432`.

[12] J. Alamos, P. Kietzmann, T. C. Schmidt, and M. Wählisch. WIP: Exploring DSME MAC for LoRa – A System Integration and First Evaluation. In *23rd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 169–172, Piscataway, NJ, USA, June 2022. IEEE. URL `https://doi.org/10.1109/WoWMoM54355.2022.00050`.

[13] G. Alderisi, G. Patti, O. Mirabella, and L. L. Bello. Simulative assessments of the ieee 802.15.4e dsme and tsch in realistic process automation scenarios. In *13th International Conference on Industrial Informatics (INDIN'15)*, pages 948–955, Piscataway, NJ, USA, 2015. IEEE.

[14] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro. Named data networking for IoT: An architectural perspective. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5, Piscataway, NJ, USA, June 2014. IEEE.

[15] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro. Information Centric Networking in IoT scenarios: The case of a smart home. In *Proc. of IEEE International Conference on Communications (ICC)*, pages 648–653, Piscataway, NJ, USA, June 2015. IEEE.

[16] Amazon Web Services. FreeRTOS Real-time operating system for microcontrollers. `https://www.freertos.org/`, last accessed 30-11-2020, 2020.

[17] Android Developers (Blog). Security "Crypto" provider deprecated in Android N. `https://android-developers.googleblog.com/2016/06/security-crypto-provider-deprecated-in.html`, last accessed 12-10-2020, 2016.

[18] M. T. H. Anik, J.-L. Danger, S. Guilley, and N. Karimi. Detecting Failures and Attacks via Digital Sensors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1315–1326, 2021.

[19] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, Aug. 2017. USENIX Association.

[20] Apache Software Foundation. Contiki-NG: The OS for Next Generation IoT Devices. `https://github.com/contiki-ng/contiki-ng`, last accessed 10-11-2020.

[21] Apache Software Foundation. Apache Mynewt. `https://mynewt.apache.org`, last accessed 07-17-2020, 2020.

[22] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`, last accessed 11-25-2020, 2020.

[23] ARM Ltd. Mbed OS. `https://www.mbed.com`, last accessed 07-17-2020, 2020.

[24] ARM Ltd. Mbed TLS. `https://tls.mbed.org`, last accessed 07-17-2020, 2020.

[25] ARM Ltd. TrustZone for Cortex-M. `https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m`, last accessed 12-10-2020, 2020.

[26] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo. Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT). *IEEE Internet of Things Journal*, 6(2):2128–2158, 2019.

[27] O. Ascigil, S. Reñé, G. Xylomenos, I. Psaras, and G. Pavlou. A Keyword-based ICN-IoT Platform. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*, pages 22–28, New York, NY, USA, September 2017. ACM.

[28] C. Ashokkumar, B. Roy, B. S. V. Mandarapu, and B. Menezes. "S-Box" Implementation of AES Is Not Side Channel Resistant. *Journal of Hardware and Systems Security*, 4: 86–97, 2019.

[29] Atmel. *Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications*. Atmel Corporation, September 2009. URL `http://www.atmel.com/images/doc8111.pdf`.

[30] *Generating Random Secrets: ATSHA204A, ATECC108A, and ATECC508A*. Atmel, September 2015. Rev. 8843B.

[31] *AT86RF233 Low Power, 2.4GHz Transceiver for ZigBee, RF4CE, IEEE 802.15.4, 6LoWPAN, and ISM Applications*. Atmel, July 2017. Rev. 8315E–MCU Wireless–07/14.

[32] 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*, pages 79–80, Piscataway, NJ, USA, 2013. IEEE Press.

[33] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch. Information Centric Networking in the IoT: Experiments with NDN in the Wild. In *Proc. of 1st ACM Conf. on Information-Centric Networking (ICN-2014)*, pages 77–86, New York, September 2014. ACM. URL `http://doi.org/10.1145/2660129.2660144`.

[34] E. Baccelli, C. Gündogan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch. RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal*, 5(6):4428–4440, December 2018. URL `http://doi.org/10.1109/JIOT.2018.2815038`.

[35] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. A. Patterson, D. Pau, J. Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav. Benchmarking TinyML Systems: Challenges and Direction. Technical Report arXiv:2003.04821, Open Archive: arXiv.org, August 2020. URL `https://arxiv.org/abs/2003.04821`.

[36] A. Banks and R. G. (Eds.). MQTT Version 3.1.1. Oasis standard, OASIS, October 2014. URL `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`.

[37] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu. Leftover Hash Lemma, Revisited. In P. Rogaway, editor, *Advances in Cryptology (CRYPTO '11)*, pages 1–20, Berlin, Heidelberg, 2011. Springer–Verlag.

[38] M. Barbareschi, E. Battista, A. Mazzeo, and N. Mazzocca. Testing 90 nm microcontroller SRAM PUF quality. In *10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'15)*, Piscataway, NJ, USA, 2015. IEEE.

[39] E. Barker. Recommendation for Key Management. Technical Report NIST SP 800-57 Part 1, National Institute of Standards and Technology, Gaithersburg, MD, US, May 2020.

[40] E. B. Barker and J. M. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Special Publication NIST SP 800-90A, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2012.

[41] E. B. Barker and J. M. Kelsey. Recommendation for Random Bit Generator (RBG) Constructions. Special Publication NIST SP 800-90C, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2016.

[42] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*, pages 43–56, New York, NY, USA, 2008. ACM.

[43] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, N. A. Heckert, J. Dray, and S. Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special Publication NIST SP 800-22, National Institute of Standards & Technology, Gaithersburg, MD, US, 2010.

[44] M. Bellare and T. Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In *EUROCRYPT '04: Advances in Cryptology*, volume 3027 of *LNCS*, pages 401–418, Berlin, Heidelberg, 2004. Springer.

[45] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[46] D. J. Bernstein. Entropy Attacks! The conventional wisdom says that hash outputs can't be controlled; the conventional wisdom is simply wrong. `https://blog.cr.yp.to/20140205-entropy.html`, last accessed 07-17-2020, 2014.

[47] D. J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In K. Kurosawa, editor, *Advances in Cryptology — ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, Berlin, Heidelberg, Germany, 2007.

[48] M. Bezunartea, R. V. Glabbeek, A. Braeken, J. Tiberghien, and K. Steenhaut. Towards Energy Efficient LoRa Multihop Networks. In *International Symposium on Local and Metropolitan Area Networks (LANMAN '19)*, pages 1–3, Piscataway, NJ, USA, 2019. IEEE.

[49] G. Bianchi, A. L. Rosa, and G. Restuccia. RIOT-AKA: cellular-like authentication over IoT devices. In *29th IEEE Int. Conf. on Network Protocols (ICNP'21)*, pages 1–6, Piscataway, NJ, USA, 2021. IEEE.

[50] D. Blackman and S. Vigna. xoshiro / xoroshiro generators and the PRNG shootout. `http://prng.di.unimi.it/`, last accessed on 04-01-2020, 2020.

[51] Bluetooth Special Interest Group. Bluetooth Core Specification. Bluetooth Specification 5.1, Bluetooth SIG, January 2019. URL `https://www.bluetooth.com/specifications/bluetooth-core-specification`.

[52] Bluetooth Special Interest Group. Mesh Profile. Bluetooth Specification 1.0.1, Bluetooth SIG, January 2019. URL `https://www.bluetooth.com/specifications/mesh-specifications/`.

[53] L. Blum, M. Blum, and M. Shub. A Simple Secure Pseudo-Random Number Generator. Technical Report UCB/ERL M82/65, EECS Department, University of California, Berkeley, 1982. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1982/28538.html`.

[54] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd annual Symp. on Foundations of Computer Science (SFCS '82)*, pages 112–117, Los Alamitos, CA, USA, 1982. IEEE Computer Society.

[55] L. Boeckmann, P. Kietzmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch. Usable Security for an IoT OS: Integrating the Zoo of Embedded Crypto Components Below a Common API. In *Proc. of Embedded Wireless Systems and Networks (EWSN'22)*, pages 84–95, New York, USA, October 2022. ACM. URL `https://dl.acm.org/doi/10.5555/3578948.3578956`.

[56] L. Boeckmann, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Poster Abstract: Offloading Crypto Processing with RIOT. In *Proc. of ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN '22), Poster Session*, pages 535–536, Piscataway, NJ, USA, May 2022. IEEE. URL `https://doi.org/10.1109/IPSN54338.2022.00068`.

[57] C. Bormann. 6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 7400, IETF, November 2014. URL `https://doi.org/10.17487/RFC7400`.

[58] C. Bormann, M. Ersue, and A. Keranen. Terminology for Constrained-Node Networks. RFC 7228, IETF, May 2014. URL `https://doi.org/10.17487/RFC7228`.

[59] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient Helper Data Key Extractor on FPGAs. In *Cryptographic Hardware and Embedded Systems - CHES 2008*, pages 181–197, Berlin, Heidelberg, 2008. Springer-Verlag.

[60] R. G. Brown. dieharder(1) - Linux man page. `https://linux.die.net/man/1/dieharder`, last accessed 12-10-2020, 2020.

[61] R. G. Brown, D. Eddelbuettel, and D. Bauer. Dieharder: A Random Number Test Suite, 2019. URL `https://webhome.phy.duke.edu/~rgb/General/dieharder.php`.

[62] J. Burke, P. Gasti, N. Nathan, and G. Tsudik. Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control and NDN. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 394–398, Piscataway, NJ, USA, 2013. IEEE.

[63] J. Burke, P. Gasti, N. Nathan, and G. Tsudik. Secure Sensing over Named Data Networking. In *13th International Symposium on Network Computing and Applications (NCA'14)*, pages 175–180, Washington, DC, USA, 2014. IEEE Computer Society.

[64] Bushing and Marcan and Segher and Sven. Console Hacking 2010 - PS3 Epic Fail. `https://fahrplan.events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`, last accessed 07-17-2020, 2010.

[65] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[66] G. Carofiglio, L. Muscariello, M. Papalini, N. Rozhnova, and X. Zeng. Leveraging ICN In-Network Control for Loss Detection and Recovery in Wireless Mobile Networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 50–59, New York, NY, USA, 2016. ACM.

[67] A. Carzaniga, M. Papalini, and A. L. Wolf. Content-based Publish/Subscribe Networking and Information-centric Networking. In *Proc. of the ACM SIGCOMM WS on Information-centric Networking (ICN '11)*, pages 56–61, New York, NY, USA, 2011. ACM.

[68] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay. A PUF-Based Secure Communication Protocol for IoT. *ACM Trans. Embed. Comput. Syst.*, 16(3), 2017.

[69] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. Ramakrishnan. COPSS: An Efficient Content Oriented Publish/Subscribe System. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'11)*, pages 99–110, Los Alamitos, CA, USA, Oct. 2011. IEEE Computer Society.

[70] Y. Chen and T. Kunz. Performance evaluation of IoT protocols under a constrained wireless access network. In *International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, pages 1–7, Piscataway, NJ, USA, 2016. IEEE.

[71] R. Chiocchetti, D. Rossi, and G. Rossini. ccnSim: An highly scalable CCN simulator. In *Proc. of IEEE International Conference on Communications (ICC'13)*, pages 2309–2314, Piscataway, NJ, USA, 2013. IEEE.

[72] N. Choudhury, R. Matam, M. Mukherjee, and J. Lloret. A Performance-to-Cost Analysis of IEEE 802.15.4 MAC With 802.15.4e MAC Modes. *IEEE Access*, 8:41936–41950, 2020.

[73] Cifra Authors. A collection of cryptographic primitives targeted at embedded use. `https://github.com/ctz/cifra`, last accessed 10-11-2020.

[74] M. Claes, V. van der Leest, and A. Braeken. Comparison of SRAM and FF PUF in 65nm Technology. In P. Laud, editor, *Information Security Technology for Applications*, pages 47–64, Berlin, Heidelberg, 2012. Springer–Verlag.

[75] T. Community. The Things Network. `https://www.thethingsnetwork.org/`, last accessed 04-12-2022, 2022.

[76] A. Compagno, M. Conti, C. Ghali, and G. Tsudik. To NACK or Not to NACK? Negative Acknowledgments in Information-Centric Networking. In *24th International Conference on Computer Communication and Networks (ICCCN'15)*, Piscataway, NJ, USA, 2015. IEEE Press.

[77] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini. An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2481–2494, 2017.

[78] H. Corrigan-Gibbs and S. Jana. Recommendations for Randomness in the Operating System, or How to Keep Evil Children out of Your Pool and Other Random Facts. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, Berkeley, CA, USA, 2015. USENIX Association.

[79] M. Cortez, S. Hamdioui, V. van der Leest, R. Maes, and G.-J. Schrijen. Adapting voltage ramp-up time for temperature noise reduction on memory-based PUFs. In *International Symposium on Hardware-Oriented Security and Trust (HOST'13)*, pages 35–40, Piscataway, NJ, USA, 2013. IEEE.

[80] J. R. Cotrim and J. ao Henrique Kleinschmidt. LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication. *Sensors*, 20(15):4273, 2020.

[81] C. Cremers, L. Garratt, S. Smyshlyaev, N. Sullivan, and C. Wood. Randomness Improvements for Security Protocols. RFC 8937, IETF, October 2020. URL `https://doi.org/10.17487/RFC8937`.

[82] CSOonline. A Critical Random Number Generator Flaw Affects Billions of IoT Devices. `https://www.csoonline.com/article/3629437/iot-devices-have-serious-security-deficiencies-due-to-bad-random-number-generation.html`, last accessed 11-10-2022, 2021.

[83] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 1999.

[84] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. Technical Report arXiv:2010.08678, Open Archive: arXiv.org, October 2020. URL `https://arxiv.org/abs/2010.08678`.

[85] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede. Ultra Low-Power Implementation of ECC on the ARM Cortex-M0+. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 1–6, New York, NY, USA, 2014. ACM.

[86] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):889–902, 2015.

[87] H. Demirhan and N. Bitirim. CryptRndTest: An R Package for Testing the Cryptographic Randomness. *The R Journal*, 8:233–247, 2016.

[88] A. Desai, A. Hevia, and Y. L. Yin. A Practice-Oriented Treatment of Pseudorandom Number Generators. In *EUROCRYPT '02: Advances in Cryptology*, volume 2332 of *LNCS*, pages 368–383, Berlin, Heidelberg, 2002. Springer.

[89] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin. Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Comput. Surv.*, 51(6):116–1 – 116–29, Jan. 2019.

[90] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM Journal on Computing*, 38 (1):97–139, 2008.

[91] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs. Security Analysis of Pseudo-random Number Generators with Input: /dev/random is not Robust. In *Proc. of the ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*, pages 647–658, New York, NY, USA, 2013. ACM.

[92] L. Dorrendorf, Z. Gutterman, and B. Pinkas. Cryptanalysis of the Windows Random Number Generator. In *Proc. of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pages 476–485, New York, NY, USA, 2007. ACM.

[93] V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. RFC 7435, IETF, December 2014. URL `https://doi.org/10.17487/RFC7435`.

[94] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of IEEE Local Computer Networks (LCN)*, pages 455–462, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[95] A. Durand, P. Gremaud, J. Pasquier, and U. Gerber. Trusted Lightweight Communication for IoT Systems Using Hardware Security. In *9th International Conference on the Internet of Things (IoT '19)*, pages 1–4, New York, NY, USA, 2019. ACM.

[96] Eclipse Foundation. IoT & Edge Developer Survey Report. `https://outreach.eclipse.foundation/iot-adoption-2019`, last accessed 03-12-2022, 2019.

[97] Eclipse Foundation. IoT & Edge Developer Survey Report. `https://outreach.eclipse.foundation/iot-edge-developer-2021`, last accessed 21-11-2022, 2021.

[98] Eclipse Foundation. IoT & Edge Developer Survey Report. `https://outreach.eclipse.foundation/iot-edge-developer-survey-2022`, last accessed 21-11-2022, 2022.

[99] G. S. Edward and S. Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proc. of the 44th Annual Design Automation Conference (DAC '07)*, pages 9–14, New York, NY, USA, 2007. ACM.

[100] I. Eichhorn, P. Koeberl, and V. van der Leest. Logically Reconfigurable PUFs: Memory-Based Secure Key Storage. In *Proc. of the 6th ACM Workshop on Scalable Trusted Computing (STC '11)*, pages 59–64, New York, NY, USA, 2011. ACM.

[101] H. E. Elbsir, M. Kassab, S. Bhiri, and M. H. Bedoui. Evaluation of LoRaWAN Class B efficiency for downlink traffic. In *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 105–110, Piscataway, NJ, USA, October 2020. IEEE.

[102] H. E. Elbsir, M. Kassab, S. Bhiri, and M. H. Bedoui. Evaluation of LoRaWAN Class B efficiency for downlink traffic. In *16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'20)*, pages 105–110, Piscataway, NJ, USA, 2020. IEEE.

[103] A. Elmangoush, R. Steinke, T. Magedanz, A. A. Corici, A. Bourreau, and A. Al-Hezmi. Application-derived communication protocol selection in M2M platforms for smart cities. In *Proc. of 18th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 76–82, Piscataway, NJ, USA, 2015. IEEE.

[104] European Telecommunications Standards Institute. Electromagnetic compatibility and Radio spectrum Matters (ERM); Short Range Devices (SRD); Radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mW; Part 1: Technical characteristics and test methods. Technical Report ETSI EN 300 220-1 V2.1.1, IEEE, Sophia Antipolis, France, Jan. 2006.

[105] A. Falcone, C. Felicetti, A. Garro, A. Rullo, and D. Saccà. PUF-Based Smart Tags for Supply Chain Management. In *16th International Conference on Availability, Reliability and Security (ARES'21)*, New York, NY, USA, 2021. ACM.

[106] R. Faraji and H. R. Naji. Adaptive Technique for Overcoming Performance Degradation Due to Aging on 6T SRAM Cells. *IEEE Transactions on Device and Materials Reliability*, 14(4):1031–1040, 2014.

[107] S. Farrell. Low-Power Wide Area Network (LPWAN) Overview. RFC 8376, IETF, May 2018. URL `https://doi.org/10.17487/RFC8376`.

[108] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley & Sons, New York, 2nd edition edition, 1971.

[109] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, Indianapolis, Indiana, USA, 2010.

[110] G. Ferre. Collision and packet loss analysis in a LoRaWAN network. In *25th European Signal Processing Conference (EUSIPCO'17)*, pages 2586–2590, Piscataway, NJ, USA, 2017. IEEE.

[111] J. Finnegan, S. Brown, and R. Farrell. Evaluating the Scalability of LoRaWAN Gateways for Class B Communication in ns-3. In *IEEE Conference on Standards for Communications and Networking (CSCN'18)*, pages 1–6, Piscataway, NJ, USA, 2018. IEEE.

[112] F. Forooghifar, A. Aminifar, and D. Atienza. Resource-Aware Distributed Epilepsy Monitoring Using Self-Awareness From Edge to Cloud. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6):1338–1350, 2019.

[113] N. Fotiou, H. Islam, D. Lagutin, T. Hakala, and G. C. Polyzos. CoAP over ICN. In *Proc. of IFIP NTMS*, pages 1–4, Piscataway, NJ, USA, 2016. IEEE.

[114] N. Fotiou, G. Xylomenos, G. C. Polyzos, H. Islam, D. Lagutin, T. Hakala, and E. Hakala. ICN Enabling CoAP Extensions for IP Based IoT Devices. In *Proc. of ACM ICN*, pages 218–219, New York, NY, USA, 2017. ACM.

[115] A. Francillon and C. Castelluccia. TinyRNG: A Cryptographic Random Number Generator for Wireless Sensors Network Nodes. In *WIOPT '07: 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops*, pages 1–7, Limassol, Cyprus, 2007. IEEE.

[116] M. Frey, C. Gündogan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, F. Shzu-Juraschek, and M. Wählisch. Security for the Industrial IoT: The Case for Information-Centric Networking. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT) (WF-IoT 2019)*, pages 424–429, Piscataway, NJ, USA, April 2019. IEEE Press. URL `http://doi.org/10.1109/WF-IoT.2019.8767183`.

[117] E. Frimpong and A. Michalas. SeCon-NG: Implementing a Lightweight Cryptographic Library Based on ECDH and ECDSA for the Development of Secure and Privacy-Preserving Protocols in Contiki-NG. In *35th Symposium on Applied Computing (SAC '20)*, pages 767–769, New York, NY, USA, 2020. ACM.

[118] N. Galbreath. *Cryptography for Internet and Database Applications: Developing Secret and Public Key Techniques with Java*. Wiley Publishing, Indianapolis, Indiana, USA, 2002.

[119] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert. Strong Machine Learning Attack Against PUFs with No Mathematical Model. In *Cryptographic Hardware and Embedded Systems (CHES'16)*, pages 391–411, Berlin, Heidelberg, 2016. Springer–Verlag.

[120] J. J. Garcia-Luna-Aceves. ADN: An Information-Centric Networking Architecture for the Internet of Things. In *Proc. of the 2nd International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 27–36, New York, NY, USA, 2017. ACM.

[121] B. Gassend, D. Clarke, van Marten Dijk, and S. Devadas. Silicon Physical Random Functions. In *Proc. of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, pages 148–160, New York, NY, USA, 2002. ACM.

[122] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, 2017.

[123] A. H. Gerez, K. Kamaraj, R. Nofal, Y. Liu, and B. Dezfouli. Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications. In *International Workshop on Signal Processing Systems (SiPS '18)*, pages 312–317, Piscataway, NJ, USA, 2018. IEEE.

[124] O. Gimenez and I. Petrov. Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN. RFC 9011, IETF, April 2021. URL `https://doi.org/10.17487/RFC9011`.

[125] E. D. Giovanni, F. Montagna, B. W. Denkinger, S. Machetti, M. P. Quiros, S. Benatti, D. Rossi, L. Benini, and D. A. Alonso. Modular Design and Optimization of Biomedical Applications for Ultra-Low Power Heterogeneous Platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3821–3832, 2020.

[126] M. Girault, R. Cohen, and M. Campana. A Generalized Birthday Attack. In *EUROCRYPT '88: Advances in Cryptology*, volume 330 of *LNCS*, pages 129–156, Berlin, Heidelberg, 1988. Springer.

[127] M. J. E. Golay. Notes on Digital Coding. *Proc. of the Institute of Radio Engineers (IRE '49)*, 37:657–657, 1949.

[128] I. Goldberg and D. Wagner. Randomness and the Netscape Browser. `https://people.eecs.berkeley.edu/~daw/papers/ddj-netscape.html`, last accessed 07-17-2020, 1996.

[129] C. Gomez, J. Crowcroft, and M. Scharf. TCP Usage Guidance in the Internet of Things (IoT). RFC 9006, IETF, March 2021. URL `https://doi.org/10.17487/RFC9006`.

[130] N. Gonzalez, A. V. D. Bossche, and T. Val. Specificities of the LoRa physical layer for the development of new ad hoc MAC layers. In *17th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now'18)*, volume 11104, pages 163–174, Cham, Switzerland, 2018. Springer.

[131] G. Grassi, D. Pesavento, L. Wang, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang. ACM HotMobile 2013 Poster: Vehicular Inter-networking via Named Data. *SIGMOBILE Mob. Comput. Commun. Rev.*, 17(3):23–24, November 2013.

[132] G. Grassi, D. Pesavento, L. Wang, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang. Vehicular Inter-Networking via Named Data. Technical Report arXiv:1310.5980, Open Archive: arXiv.org, October 2013.

[133] M. Green and M. Smith. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security and Privacy*, 14(5):40–46, 2016.

[134] S. Greenland, S. J. Senn, K. J. Rothman, J. B. Carlin, C. Poole, S. N. Goodman, and D. G. Altman. Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations. *European Journal of Epidemiology*, 31(4):337–350, 2016.

[135] M. Gritter and D. R. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proc. USITS'01*, pages 4–4, Berkeley, CA, USA, 2001. USENIX Association.

[136] T. C. Group. Trusted Platform Module Library, Part 1: Architecture, Level 00 Revision 1.59. Technical report, TCG Published, November 2019. URL `https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf`.

[137] C. Gu, C.-H. Chang, W. Liu, N. Hanley, J. Miskelly, and M. O'Neill. A large-scale comprehensive evaluation of single-slice ring oscillator and PicoPUF bit cells on 28-nm Xilinx FPGAs. *Journal of Cryptographic Engineering*, 11(3):227–238, 2021.

[138] J. Guajardo, S. S.Kumar, G.-J. Schrijen, and P. Tuyls. FPGA Intrinsic PUFs and Their Use for IP Protection. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems (CHES'07)*, pages 63–80, Berlin, Heidelberg, 2007. Springer–Verlag.

[139] U. Guin, W. Wang, C. Harper, and A. D. Singh. Detecting Recycled SoCs by Exploiting Aging Induced Biases in Memory Cells. In *International Symposium on Hardware Oriented Security and Trust (HOST'19)*, pages 72–80, Piscataway, NJ, USA, 2019. IEEE.

[140] C. Gündogan. *Information-centric Networking for the Constrained Internet of Things.* Doctoral dissertation, Department of Mathematics and Computer Science, Freie Universität Berlin, July 2022. URL `http://doi.org/10.17169/refubium-35327`.

[141] C. Gündogan, P. Kietzmann, T. C. Schmidt, M. Lenders, H. Petersen, M. Wählisch, M. Frey, and F. Shzu-Juraschek. Information-Centric Networking for the Industrial IoT. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN), Demo Session*, pages 214–215, New York, NY, USA, September 2017. ACM. URL https://doi.org/10.1145/3125719.3132099.

[142] C. Gündogan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN)*, pages 159–171, New York, NY, USA, September 2018. ACM. URL https://doi.org/10.1145/3267955.3267967.

[143] C. Gündogan, P. Kietzmann, T. C. Schmidt, M. Lenders, H. Petersen, M. Wählisch, M. Frey, and F. Shzu-Juraschek. Demo: Seamless Producer Mobility for the Industrial Information-Centric Internet. In *Proc. of 16th ACM International Conference on Mobile Systems, Applications (MobiSys), Demo Session*, New York, NY, USA, June 2018. ACM. URL https://doi.org/10.1145/3210240.3211114.

[144] C. Gündogan, P. Kietzmann, T. C. Schmidt, M. Lenders, H. Petersen, M. Wählisch, M. Frey, and F. Shzu-Juraschek. Resilient Machine-to-Machine Communication for an Information-centric Industrial IoT. In *Proc. of the 43rd Annual IEEE Conference on Local Computer Networks (LCN'18, Demo Session)*, Piscataway, NJ, USA, October 2018. IEEE Press.

[145] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. HoPP: Publish–Subscribe for the Constrained IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN), Demo Session*, pages 216–217, New York, NY, USA, September 2018. ACM. URL https://doi.org/10.1145/3267955.3269020.

[146] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. In *Proc. of the 43rd IEEE Conference on Local Computer Networks (LCN)*, pages 331–334, Piscataway, NJ, USA, Oct. 2018. IEEE Press. URL http://doi.org/10.1109/LCN.2018.8638030.

[147] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. ICN-LoWPAN: Header Compression for the Constrained IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN), Poster Session*, pages 184–185, New York, NY, USA, September 2018. ACM. URL https://doi.org/10.1145/3267955.3269006.

[148] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. ICNLoWPAN – Named-Data Networking in Low Power IoT Networks. In *Proc. of 18th IFIP Networking Conference*, pages 1–9, Piscataway, NJ, USA, May 2019. IEEE Press. URL http://doi.org/10.23919/IFIPNetworking.2019.8816850.

[149] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Your Message Rescues Me: Enhancing NDN Communication Quality in Disaster Scenarios. In *Proc. of 6th ACM Conference on Information-Centric Networking (ICN), Demo Session*, pages 173–174, New York, September 2019. ACM. URL https://doi.org/10.1145/3357150.3357414.

[150] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch. IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison. In *Proc. of 19th IFIP Networking Conference*, pages 19–27, Piscataway, NJ, USA, June 2020. IEEE Press. URL https://ieeexplore.ieee.org/document/9142731.

[151] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Designing a LoWPAN convergence layer for the Information Centric Internet of Things. *Computer Communications*, 164(1):114–123, December 2020. ISSN 0140-3664. URL https://doi.org/10.1016/j.comcom.2020.10.002.

[152] C. Gündogan, J. Pfender, P. Kietzmann, T. C. Schmidt, and M. Wählisch. On the Impact of QoS Management in an Information-centric Internet of Things. *Computer Communications*, 154:160–172, March 2020. ISSN 0140-3664. URL https://doi.org/10.1016/j.comcom.2020.02.046.

[153] C. Gündogan, P. Kietzmann, M. S. Lenders, H. Petersen, M. Frey, T. C. Schmidt, F. Shzu-Juraschek, and M. Wählisch. The Impact of Networking Protocols on Massive M2M Communication in the Industrial IoT. *IEEE Transactions on Network and Service Management (TNSM)*, 18(4):4814–4828, Dec. 2021. URL https://doi.org/10.1109/TNSM.2021.3089549.

[154] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Information-Centric Networking for the Industrial Internet of Things. In N. H. Mahmood, N. Marchenko, M. Gidlund, and P. Popovski, editors, *Wireless Networks and Industrial IoT*, pages 171–189. Springer, Februar 2021. ISBN 978-3-030-51473-0. URL https://doi.org/10.1007/978-3-030-51473-0_9.

[155] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch. Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions. *IEEE Transactions on Network and Service Management (TNSM)*, 19(1):538–553, March 2022. URL https://doi.org/10.1109/TNSM.2021.3099902.

[156] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. A Mobility-compliant Publish Subscribe System for an Information Centric Internet of Things. *Computer Networks*, 203(108656):1–14, February 2022. ISSN 1389-1286. URL https://doi.org/10.1016/j.comnet.2021.108656.

[157] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132, Berlin, Heidelberg, Germany, 2004. Springer.

[158] P. Gutmann. Software Generation of Practically Strong Random Numbers. In *SSYM '98: Proc. of 7th USENIX Security Symposium*, pages 19–19, Berkeley, CA, USA, 1998. USENIX Association.

[159] Z. Guttermann, B. Pinkas, and T. Reinman. Analysis of the Linux Random Number Generator. In *Symposium on Security and Privacy (S&P '06)*, pages 371–385, Berkeley, CA, USA, 2006. IEEE.

[160] C. Gündoğan, T. C. Schmidt, and M. Wählisch. Publish-Subscribe Deployment Option for NDN in the Constrained Internet of Things. Internet-Draft – work in progress 01, IETF, July 2017. URL `https://datatracker.ietf.org/doc/html/draft-gundogan-icnrg-pub-iot-01`.

[161] C. Gündoğan, T. C. Schmidt, M. Wählisch, C. Scherb, C. Marxer, and C. Tschudin. ICN Adaptation to LowPAN Networks (ICN LoWPAN). Internet-Draft – work in progress 02, IETF, July 2018. URL `https://datatracker.ietf.org/doc/html/draft-gundogan-icnrg-ccnlowpan-02`.

[162] O. Hahm, C. Adjih, E. Baccelli, T. C. Schmidt, and M. Wählisch. ICN over TSCH: Potentials for Link-Layer Adaptation in the IoT. In *Proc. of 3rd ACM Conf. on Information-Centric Networking (ICN 2016), Poster Session*, pages 195—196, New York, NY, USA, September 2016. ACM. URL `http://doi.org/10.1145/2984356.2985226`.

[163] O. Hahm, C. Adjih, E. Baccelli, T. C. Schmidt, and M. Wählisch. Designing Time Slotted Channel Hopping and Information-Centric Networking for IoT. In *Proc. of 8th IFIP International Conference on New Technologies, Mobility & Security (NTMS)*, Piscataway, NJ, USA, November 2016. IEEE Press.

[164] O. Hahm, E. Baccelli, T. C. Schmidt, M. Wählisch, C. Adjih, and L. Massoulié. Low-power Internet of Things with NDN and Cooperative Caching. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*, pages 98–108, New York, NY, USA, September 2017. ACM.

[165] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[166] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, IETF, November 1998. URL `https://doi.org/10.17487/RFC2409`.

[167] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, IETF, September 2015. URL `https://doi.org/10.17487/RFC7641`.

[168] M. Haubro, C. Orfanidis, G. Oikonomou, and X. Fafoutis. TSCH-over-LoRA: long range and reliable IPv6 multi-hop networks for the internet of things. *Internet Technology Letters*, 3(4):e165, 2020.

[169] Y. He, D. Li, Z. Yu, and K. Yang. ASCH-PUF: A "Zero" Bit Error Rate CMOS Physically Unclonable Function With Dual-Mode Low-Cost Stabilization. *IEEE Journal of Solid-State Circuits*, pages 1–11, early access, Jan. 2023.

[170] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert. Cloning Physically Unclonable Functions. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST '13)*, pages 1–6, Piscataway, NJ, USA, June 2013. IEEE.

[171] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley. Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs. In *International Symposium on Hardware-Oriented Security and Trust (HOST'18)*, pages 126–133, Piscataway, NJ, USA, 2018. IEEE.

[172] M. Hiller, L. Kürzinger, and G. Sigl. Review of error correction for PUFs and evaluation on state-of-the-art FPGAs. *Journal of Cryptographic Engineering*, 10(3):229–247, 2020.

[173] D. E. Holcomb, W. P. Burleson, and K. Fu. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*, 58 (9):1198–1210, 2009.

[174] IEEE 802.15 Working Group. IEEE Standard for Low-Rate Wireless Networks. Technical Report IEEE Std 802.15.4™–2015 (Revision of IEEE Std 802.15.4-2011), IEEE, New York, NY, USA, 2016.

[175] M. Iglesias-Urkia, A. Orive, and A. Urbieta. Analysis of CoAP Implementations for Industrial Internet of Things: A Survey. *Procedia Computer Science*, 109:188–195, 2017.

[176] INET Authors. INET Framework - An open-source OMNeT++ model suite for wired, wireless and mobile networks. `https://inet.omnetpp.org/`, last accessed 06-04-2021, 2021.

[177] Intel Corporation. TinyCrypt Cryptographic Library. `https://github.com/intel/tinycrypt`, last accessed 07-17-2020, 2017.

[178] International Society of Automation. Wireless Systems for Industrial Automation: Process Control and Related Applications. Technical Report Standard ISA-100.11a-2011, ISA, 2011.

[179] H. Islam, D. Lagutin, and N. Fotiou. Observing IoT Resources over ICN. In *Proc. of IFIP Networking Workshop on Information-Centric Fog Computing*, pages 1–8, Piscataway, NJ, USA, 2017. IEEE.

[180] V. Jacobson, D. K. Smetters, J. D. Thornton, and M. F. Plass. Networking Named Content. In *5th Int. Conf. on emerging Networking Experiments and Technologies (ACM CoNEXT'09)*, pages 1–12, New York, NY, USA, Dec. 2009. ACM.

[181] José Quevedo and Rui Ferreira and Carlos Guimarães and Rui L. Aguiar and Daniel Corujo. Internet of things discovery in interoperable information centric and ip networks. *Internet Technology Letters*, 1:1–6, 2017.

[182] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. In *Proc. of the 6th ACM Conference on Computer and Communications Security (CCS '99)*, pages 28–36, New York, NY, USA, 1999. ACM.

[183] D. Kaplan, S. Kedmi, R. Hay, and A. Dayan. Attacking the Linux PRNG on Android: Weaknesses in Seeding of Entropic Pools and Low Boot-Time Entropy. In *8th USENIX Conference on Offensive Technologies (WOOT '14)*, Berkeley, CA, USA, 2014. USENIX Association.

[184] T. Karunathilake, A. Udugama, and A. Förster. LoRa-DuCy: Duty Cycling for LoRa-Enabled Internet of Things Devices. In *12th International Conference on Ubiquitous and Future Networks (ICUFN '21)*, pages 283–288, Piscataway, NJ, USA, 2021. IEEE.

[185] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, and I. V. C. Wachsmann. PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems (CHES '12)*, pages 283–301, Berlin, Heidelberg, 2012. Springer–Verlag.

[186] F. Kauer, M. Köstler, and V. Turau. Reliable Wireless Multi-Hop Networks with Decentralized Slot Management: An Analysis of IEEE 802.15.4 DSME. Technical Report arXiv:1806.10521, Open Archive: arXiv.org, June 2018.

[187] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic Attacks on Pseudorandom Number Generators. In *FSE '98: Proceedings of the 5th International Workshop on Fast Software Encryption*, pages 168–188, London, UK, UK, 1998. Springer-Verlag.

[188] J. Kelsey, B. Schneier, and N. Ferguson. Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. In *6th Annual Workshop on Selected Areas in Cryptography*, pages 13–33, Berlin, Heidelberg, 1999. Springer.

[189] Ken MacKay. micro-ecc. `http://kmackay.ca/micro-ecc/`, last accessed 10-11-2020.

[190] P. Kietzmann. RIOT - Betriebssystem für das IoT. In *Informationstechnologie von A-Z*. Weka Media GmbH & Co. KG, Kissing, Germany, 2017. ISBN 978-3-8245-8190-0.

[191] P. Kietzmann, M. Landsmann, T. C. Schmidt, H. Petersen, M. Lenders, and M. Wählisch. Leistungsmessung eines modularen Netzwerk-Stacks für das IoT-Betriebssystem RIOT. In *Proc. of the 14. GI/ITG KuVS Fachgespräch Sensornetze (FGSN2015)*, pages 19–22, Erlangen-Nürnberg, Germany, Sep 2015. Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science.

[192] P. Kietzmann, T. C. Schmidt, and M. Wählisch. RIOT - das freundliche Echtzeitbetriebssystem für das IoT. In *Internet der Dinge*, pages 43–52, Berlin, Nov. 2016. Springer Vieweg. URL `https://www.doi.org/10.1007/978-3-662-53443-4_5`.

[193] P. Kietzmann, C. Gündogan, T. C. Schmidt, O. Hahm, and M. Wählisch. The Need for a Name to MAC Address Mapping in NDN: Towards Quantifying the Resource Gain. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*, pages 36–42, New York, NY, USA, September 2017. ACM. URL `https://dl.acm.org/doi/10.1145/3125719.3125737`.

[194] P. Kietzmann, C. Gündogan, T. C. Schmidt, and M. Wählisch. A PUF Seed Generator for RIOT: Introducing Crypto-Fundamentals to the Wild. In *Proc. of 16th ACM International Conference on Mobile Systems, Applications (MobiSys), Poster Session*, New York, NY, USA, June 2018. ACM. URL `https://doi.org/10.1145/3210240.3210805`.

[195] P. Kietzmann, D. Kutscher, T. C. Schmidt, and M. Wählisch. Long-Range IoT: Is LoRaWAN an Option for ICN? In *Proc. of the 7th ACM Conference on Information-Centric Networking (ICN)*, pages 152–154, New York, NY, USA, 2020. ACM. URL `https://doi.org/10.1145/3405656.3420228`.

[196] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch. A Performance Study of Crypto-Hardware in the Low-end IoT. In *Proc. of Embedded Wireless Systems and Networks (EWSN'21)*, New York, USA, February 2021. ACM. URL `https://dl.acm.org/doi/10.5555/3451271.3451279`.

[197] P. Kietzmann, J. Alamos, D. Kutscher, T. C. Schmidt, and M. Wählisch. Delay-Tolerant ICN and Its Application to LoRa. In *Proc. of 9th ACM Conference on Information-Centric Networking (ICN)*, pages 125–136, New York, September 2022. ACM. URL `https://doi.org/10.1145/3517212.3558081`.

[198] P. Kietzmann, J. Alamos, D. Kutscher, T. C. Schmidt, and M. Wählisch. Long-Range ICN for the IoT: Exploring a LoRa System Design. In *Proc. of 21th IFIP Networking Conference*, pages 1–9, Piscataway, NJ, USA, June 2022. IEEE Press. URL `https://doi.org/10.23919/IFIPNetworking55013.2022.9829792`.

[199] P. Kietzmann, T. C. Schmidt, and M. Wählisch. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM Comput. Surv.*, 54(6):112:1–112:38, July 2022. URL `https://dl.acm.org/doi/10.1145/3453159`.

[200] P. Kietzmann, T. C. Schmidt, and M. Wählisch. PUF for the Commons: Enhancing Embedded Security on the OS Level. *IEEE Transactions on Dependable and Secure Computing*, 2023. URL `http://doi.org/10.1109/TDSC.2023.3300368`.

[201] W. Killmann and W. Schindler. A proposal for: Functionality classes for random number generators. Technical Report AIS 20 / AIS 31, BSI, Bonn, Germany, 2011.

[202] K. H. Kim, J. Choe, S. Y. Kim, N. Kim, and S. Hong. Speeding up Elliptic Curve Scalar Multiplication without Precomputation. *IACR Cryptol. ePrint Arch.*, (Report 2017/669), 2017.

[203] Y.-S. Kim and G. Kim. A Performance Analysis of Lightweight Cryptography Algorithm for Data Privacy in IoT Devices. In *International Conference on Information and Communication Technology Convergence (ICTC '18)*, pages 936–938. IEEE, 2018.

[204] R. King and R. Sugiyama. A New Remote Communications Link to Reduce Residential PV Solar Costs. Technical Report DE-EE0007592, U.S. Department of Energy Office of Scientific and Technical Information, 2017.

[205] D. E. Knuth. *The Art of Computer Programming (Second Edition)*. Addison Wesley, Reading, MA, USA, 2009. ISBN 0-201-89685-0.

[206] P. Koeberl, J. Li, A. Rajan, and W. Wu. Entropy loss in PUF-based key generation schemes: The repetition code pitfall. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST '14)*, pages 44–49, Piscataway, NJ, USA, 2014. IEEE.

[207] L. Kohnfelder and P. Garg. The threats to our products. Technical report, Microsoft, 1999. URL `https://adam.shostack.org/microsoft/The-Threats-To-Our-Products.docx`.

[208] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and beyond) Network Architecture. *SIGCOMM Computer Communications Review*, 37(4):181–192, 2007.

[209] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum. A Proof of Concept SRAM-based Physically Unclonable Function (PUF) Key Generation Mechanism for IoT Devices. In *Workshop on Security Trust and Privacy in Emerging Cyber-Physical Systems (SECON'19)*, Piscataway, NJ, USA, 2019. IEEE.

[210] K. Krentz, C. Meinel, and H. Graupner. Secure self-seeding with power-up SRAM states. In *ISCC '17: Symposium on Computers and Communications*, pages 1251–1256, Heraklion, Greece, 2017. IEEE.

[211] M. Król, K. Habak, D. Oran, D. Kutscher, and I. Psaras. RICE: Remote Method Invocation in ICN. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, ICN '18, pages 1–11, New York, NY, USA, 2018. ACM. ISBN 9781450359597.

[212] C. P. Kruger and G. P. Hancke. Benchmarking Internet of things devices. In *Proc. of 12th IEEE International Conf on Industrial Informatics (INDIN)*, pages 611–616, Piscataway, NJ, USA, 2014. IEEE.

[213] M. Kuai, X. Hong, and Q. Yu. Delay-tolerant forwarding strategy for named data networking in vehicular environment. *International Journal of Ad Hoc and Ubiquitous Computing*, 31(1), 2019.

[214] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric. All Things Considered: An Analysis of IoT Devices on Home Networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1169–1185, Santa Clara, CA, Aug. 2019. USENIX Association. ISBN 978-1-939133-06-9.

[215] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler. Performant TCP for Low-Power Wireless Networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 911–932, Santa Clara, CA, Feb. 2020. USENIX Association.

[216] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Impact of NBTI on SRAM read stability and design for reliability. In *7th International Symposium on Quality Electronic Design (ISQED'06)*, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[217] R. Kumari and R. L. Ujjwal. Name Data Networking for Interplanetary Internet: An Architectural Perspective. *International Journal of Research in Advent Technology*, 7(5): 436–441, 2019.

[218] H. Kurunathan, R. Severino, A. Koubaa, and E. Tovar. Symphony: Routing Aware Scheduling for DSME Networks. *SIGBED Review*, 16(4):26–31, January 2020.

[219] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. Schmidt, and M. Waehlisch. Information-Centric Networking (ICN) Research Challenges. RFC 7927, IETF, July 2016. URL `https://doi.org/10.17487/RFC7927`.

[220] C. Lachner and S. Dustdar. A Performance Evaluation of Data Protection Mechanisms for Resource Constrained IoT Devices. In *International Conference on Fog Computing (ICFC '19)*, pages 47–52, Piscataway, NJ, USA, 2019. IEEE.

[221] M. Landsmann, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Demo: Topological Robustness of RPL with TRAIL. In *Proc. of Embedded Wireless Systems and Networks*

*(EWSN '16), Demonstration*, pages 219–220, New York, NY, USA, Feb. 2016. ACM. URL https://dl.acm.org/doi/10.5555/2893711.2893742.

[222] L. Lanzieri, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Poster Abstract: Third Party Authorization of LwM2M Clients. In *Proc. of ACM/IEEE Int. Conf. on Internet of Things Design and Implementation (IoTDI '21)*, pages 263–264, New York, NY, USA, May 2021. ACM. URL https://doi.org/10.1145/3450268.3453512.

[223] L. Lanzieri, P. Kietzmann, T. C. Schmidt, and M. Wählisch. Secure and Authorized Client-to-Client Communication for LwM2M. In *Proc. of ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN '22)*, pages 158–170, Piscataway, NJ, USA, May 2022. IEEE. URL https://doi.org/10.1109/IPSN54338.2022.00020.

[224] L. Lanzieri, P. Kietzmann, G. Fey, H. Schlarb, and T. C. Schmidt. Ageing Analysis of Embedded SRAM on a Large-Scale Testbed Using Machine Learning. In *Proc. of 26th Euromicro Conference on Digital System Design (DSD)*, pages 335–342. IEEE, September 2023. URL https://doi.org/10.1109/DSD60849.2023.00054.

[225] P. L'Ecuyer and R. Simard. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Trans. Math. Softw.*, 33(4):1–40, 2007.

[226] P. L'Ecuyer and R. Simard. TestU01 A Software Library in ANSI C for Empirical Testing of Random Number Generators. User's guide, compact version. Technical report, Department of Computer Science and Operations Research, University of Montreal, Montreal, Canada, 2013.

[227] H.-C. Lee and K.-H. Ke. Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation. *IEEE Transactions on Instrumentation and Measurement*, 67(9):2177–2187, 2018.

[228] J. Lee, D.-W. Jee, and D. Jeon. Power-up control techniques for reliable SRAM PUF. *IEICE Electronics Express*, 16(13), 2019.

[229] D. H. Lehmer. Mathematical Methods in Large-scale Computing Units. In *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, pages 141–146, Cambridge, MA, US, 1951. Harvard University Press.

[230] M. Lenders, P. Kietzmann, O. Hahm, H. Petersen, C. Gündogan, E. Baccelli, K. Schleiser, T. C. Schmidt, and M. Wählisch. Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things. Technical Report arXiv:1801.02833, Open Archive: arXiv.org, January 2018. URL https://arxiv.org/abs/1801.02833.

[231] M. S. Lenders, C. Gündogan, T. C. Schmidt, and M. Wählisch. Connecting the Dots: Selective Fragment Recovery in ICNLoWPAN. In *Proc. of 7th ACM Conference on Information-Centric Networking (ICN)*, pages 70–76, New York, September 2020. ACM. URL `https://doi.org/10.1145/3405656.3418719`.

[232] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. `https://eprint.iacr.org/2012/064`, last accessed 07-17-2020, 2012.

[233] L. Leonardi, L. L. Bello, F. Battaglia, and G. Patti. Comparative Assessment of the LoRaWAN Medium Access Control Protocols for IoT: Does Listen before Talk Perform Better than ALOHA? *Electronics*, 9(4):553, 2020.

[234] C. Lerche, K. Hartke, and M. Kovatsch. Industry adoption of the Internet of Things: A constrained application protocol survey. In *Proc. 17th IEEE International Conf on Emerging Technologies & Factory Automation (ETFA)*, pages 1–6, Piscataway, NJ, USA, 2012. IEEE.

[235] T. Li, Z. Kong, and L. Zhang. Supporting Delay Tolerant Networking: A Comparative Study of Epidemic Routing and NDN. In *International Conference on Communications Workshops (ICC'20 Workshop)*, Piscataway, NJ, USA, 2020. IEEE Press.

[236] Y. Li, A. Afanasyev, J. Shi, H. Zhang, Z. Zhang, T. Li, E. Lu, B. Zhang, L. Wang, and L. Zhang. NDN Automatic Prefix Propagation. Technical Report NDN-0045, NDN, March 2018.

[237] J. C. Liando, A. Gamage, A. W. Tengourtius, and M. Li. Known and Unknown Facts of LoRa: Experiences from a Large-Scale Measurement Study. *Transactions on Sensor Networks (TOSN)*, 15(2):16, Feb. 2019. URL `https://doi.org/10.1145/3293534`.

[238] T. Liang, Z. Xia, G. Tang, Y. Zhang, and B. Zhang. NDN in Large LEO Satellite Constellations: A Case of Consumer Mobility Support. In *Proceedings of the 8th ACM Conference on Information-Centric Networking*, New York, NY, USA, 2021. ACM.

[239] Libsodium Community. A modern, portable, easy to use crypto library. `https://github.com/jedisct1/libsodium`, last accessed 12-10-2020, 2020.

[240] C. H. Lim and P. J. Lee. More Flexible Exponentiation with Precomputation. In *Advances in Cryptology (CRYPTO'94)*, pages 95–107, Berlin, Heidelberg, 1994. Springer.

[241] J. D. C. Little. A Proof for the Queuing Formula: L = $\lambda$W. *Operations Research*, 9(3): 383–387, 1961.

[242] H. Liu, W. Liu, Z. Lu, Q. Tong, and Z. Liu. Methods for Estimating the Convergence of Inter-Chip Min-Entropy of SRAM PUFs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(2):593–605, 2018.

[243] Y. Liu, L. Njilla, A. Dowling, and W. Du. Empowering named data networks for ad-hoc long-range communication. In *Wireless and Optical Communications Conference (WOCC'20)*, pages 1–6, Piscataway, NJ, USA, 2020. IEEE.

[244] P. Longa and C. Gebotys. Novel Precomputation Schemes for Elliptic Curve Cryptosystems. In *Applied Cryptography and Network Security (ACNS'09)*, pages 71–88, Berlin, Heidelberg, 2009. Springer.

[245] LoRa Alliance – Technical Committee. Lorawan 1.1 specification. Technical report, LoRa Alliance, Oct. 2017. URL `https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf`.

[246] A. Ludovici, P. Moreno, and A. Calveras. TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS. *J. Sensor and Actuator Networks*, 2(2):288–315, 2013.

[247] J. Mades, G. Ebelt, B. Janjic, F. Lauer, C. C. Rheinländer, and N. Wehn. TLS-Level Security for Low Power Industrial IoT Network Infrastructures. In *Design, Automation Test in Europe Conference Exhibition (DATE '20)*, pages 1720–1721, Piscataway, NJ, USA, 2020. IEEE.

[248] R. Maes and V. van der Leest. Countering the effects of silicon aging on SRAM PUFs. In *International Symposium on Hardware-Oriented Security and Trust (HOST'14)*, pages 148–153, Piscataway, NJ, USA, 2014. IEEE.

[249] R. Maes, A. V. Herrewege, and I. Verbauwhede. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems (CHES '12)*, pages 302–319, Berlin, Heidelberg, 2012. Springer–Verlag.

[250] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems. Secure key generation from biased PUFs: extended version. *Journal of Cryptographic Engineering*, 6(2):121–137, 2016.

[251] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. A large scale characterization of RO-PUF. In *International Symposium on Hardware-Oriented Security and Trust (HOST'10)*, pages 94–99, Piscataway, NJ, USA, 2010. IEEE.

[252] G. Marsaglia. Random Numbers Fall Mainly in the Planes. *Proc. of the National Academy of Sciences*, 61(1):25–28, 1968.

[253] G. Marsaglia. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. `https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard`, last accessed 07-17-2020, 1995. Originally published in `https://stat.fsu.edu/pub/diehard`.

[254] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software, Articles*, 8(14):1–6, 2003. URL `https://www.jstatsoft.org/v008/i14`.

[255] Marten van Hulst. Anchoring TrustZone with SRAM PUF. `https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/anchoring-trustzone-with-sram-puf`, last accessed 09-29-2021, 2019.

[256] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A Scalable Library for Pseudo-random Number Generation. *ACM Trans. Math. Softw.*, 26(3):436–461, 2000.

[257] J. L. Massey. Guessing and entropy. In *Proceedings of IEEE International Symposium on Information Theory (ISIT'94)*, page 204, 1994.

[258] B. Mathieu, C. Westphal, and P. Truong. Towards the usage of ccn for iot networks. In *Internet of Things (IoT) in 5G Mobile Technologies*, pages 3–24. Springer, Cham, Switzerland, 2016.

[259] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator. *ACM Trans. Model. Comput. Simul.*, 8(1): 3–30, 1998.

[260] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young. A PUF taxonomy. *Applied Physics Reviews*, 6(1):011303, 2019.

[261] *ATECC508A CryptoAuthentication Device Complete Data Sheet.* Microchip, December 2017. Rev. A.

[262] Microchip Technology. CryptoAuthLib – Microchip CryptoAuthentication Library. `https://github.com/MicrochipTech/cryptoauthlib`, last accessed 12-10-2020, 2020.

[263] K. Mikhaylov, J. Petäjäjärvi, and A. Pouttu. Effect of Downlink Traffic on Performance of LoRaWAN LPWA Networks: Empirical Study. In *29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'18)*, Piscataway, NJ, USA, 2018. IEEE.

[264] K. Mikhaylov, R. Fujdiak, A. Pouttu, V. Miroslav, L. Malina, and P. Mlynek. Energy Attack in LoRaWAN: Experimental Validation. In *14th International Conference on Availability, Reliability and Security (ARES '19)*, pages 1–6, New York, NY, USA, 2019. ACM.

[265] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and J. Zuniga. SCHC: Generic Framework for Static Context Header Compression and Fragmentation. RFC 8724, IETF, April 2020. URL `https://doi.org/10.17487/RFC8724`.

[266] A. Minaburo, L. Toutain, and R. Andreasen. Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP). RFC 8824, IETF, June 2021. URL `https://doi.org/10.17487/RFC8824`.

[267] K. Mindermann, P. Keck, and S. Wagner. How Usable Are Rust Cryptography APIs? In *International Conference on Software Quality, Reliability and Security (QRS '18)*, pages 143–.154, Los Alamitos, CA, USA, 2018. IEEE Computer Society.

[268] Mininet Project Contributors. Mininet - An Instant Virtual Network on your Laptop (or other PC). `http://www.mininet.org/`, last accessed 09-06-2022, 2022.

[269] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis (Second Edition)*. Cambridge University Press, Cambridge, MA, USA, 2017. ISBN 110715488X, 9781107154889.

[270] I. Moiseenko, L. Wang, and L. Zhang. Consumer / Producer Communication with Application Level Framing in Named Data Networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, ICN '15, pages 99–108, New York, NY, USA, 2015. ACM. ISBN 9781450338554.

[271] Monocypher Authors. Boring crypto that simply works. `https://monocypher.org/`, last accessed 12-10-2020, 2020.

[272] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, September 2007. URL `https://doi.org/10.17487/RFC4944`.

[273] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[274] M. Mosko, I. Solis, and C. Wood. Content-Centric Networking (CCNx) Semantics. RFC 8569, IETF, July 2019. URL `https://doi.org/10.17487/RFC8569`.

[275] M. Mössinger, B. Petschkuhn, J. Bauer, R. C. Staudemeyer, M. Wójcik, and H. C. Pöhls. Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. In *17th Intern. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'16)*, pages 1–6. IEEE, 2016.

[276] P. S. Munoz, N. Tran, B. Craig, B. Dezfouli, and Y. Liu. Analyzing the Resource Utilization of AES Encryption on IoT Devices. In *Asia-Pacific Signal and Information Processing*

*Association Annual Summit and Conference (APSIPA ASC'18)*, pages 1200–1207, Piscataway, NJ, USA, 2018. IEEE.

[277] Musl C Authors. musl libc a new libc striving to be fast, simple, lightweight, free, and correct. `https://wiki.musl-libc.org/`, last accessed 12-10-2020, 2020.

[278] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez. IPv6 over BLUETOOTH(R) Low Energy. RFC 7668, IETF, October 2015. URL `https://doi.org/10.17487/RFC7668`.

[279] NIST. Digital Signature Standard. Federal Information Processing Standards 186–1, National Institute of Standards & Technology, Gaithersburg, MD, US, December 1998.

[280] NIST. Security Requirements for Cryptographic Modules. Federal Information Processing Standards 140–1, National Institute of Standards & Technology, Gaithersburg, MD, US, January 2001.

[281] NIST. Security Requirements for Cryptographic Modules. Federal Information Processing Standards 140–2, National Institute of Standards & Technology, Gaithersburg, MD, US, May 2002. Supersedes FIPS 140–1.

[282] NIST. Standards for Security Categorization of Federal Information and Information Systems. Technical Report FIPS-199, National Institute of Standards and Technology, Gaithersburg, MD, US, February 2004.

[283] NIST. Digital Signature Standard. Federal Information Processing Standards 186–3, National Institute of Standards & Technology, Gaithersburg, MD, US, June 2009. Supersedes FIPS 186–2.

[284] NIST. Digital Signature Standard. Federal Information Processing Standard 186–4, National Institute of Standards & Technology, Gaithersburg, MD, US, July 2013.

[285] R. A. Nofal, N. Tran, C. Garcia, Y. Liu, and B. Dezfouli. A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms. In *22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '19)*, pages 61–70, New York, NY, USA, 2019. ACM.

[286] K. Nohl, D. Evans, S. Starbug, and H. Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *17th Conference on Security Symposium (SS '08)*, pages 185–193, San Jose, CA, USA, 2008. USENIX Association.

[287] *nRF52840 Product Specification*. Nordic Semiconductor, March 2018. Version 1.0.

[288] *MKW2xD Reference Manual*. NXP, May 2016. Rev. 3.

[289] S. Y. Oh, D. Lau, and M. Gerla. Content Centric Networking in tactical and emergency MANETs. In *2010 IFIP Wireless Days*, pages 1–5, Piscataway, NJ, USA, Oct 2010. IEEE.

[290] M. O'Kennedy, T. Niesler, R. Wolhuter, and N. Mitton. Practical evaluation of carrier sensing for a LoRa wildlife monitoring network. In *Proc. of 19th IFIP Networking Conference*, pages 10–18, Piscataway, NJ, USA, June 2020. IEEE Press.

[291] K. Okeya and T. Takagi. The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks. In M. Joye, editor, *Topics in Cryptology — CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 328–343. Springer, Berlin, Heidelberg, Germany, 2003.

[292] N. Ollrogge, P. Kietzmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch. First Steps Towards FIDO2 for the Internet of Things. In *Proc. of the 19. GI/ITG KuVS Fachgespräch Sensornetze (FGSN'22)*, pages 29–32, Berlin, Germany, Sep 2022. HTW Berlin. URL `https://doi.org/10.34702/mncp-qb18`.

[293] OpenISA Community. OpenISA VEGAboard. `https://open-isa.org/`, last accessed 12-10-2020, 2020.

[294] D. R. Oran and D. Kutscher. Reflexive Forwarding for CCNx and NDN Protocols. Internet-Draft – work in progress 01, IETF, April 2020. URL `https://datatracker.ietf.org/doc/html/draft-oran-icnrg-reflexive-forwarding-01`.

[295] D. R. Oran and D. Kutscher. Reflexive Forwarding for CCNx and NDN Protocols. Internet-Draft – work in progress 06, IETF, September 2023. URL `https://datatracker.ietf.org/doc/html/draft-oran-icnrg-reflexive-forwarding-06`.

[296] C. Orfanidis, L. M. Feeney, M. Jacobsson, and P. Gunningberg. Cross-Technology Clear Channel Assessment for Low-Power Wide Area Networks. In *16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS'19)*, pages 199–207, Washington, DC, USA, 2019. IEEE Computer Society.

[297] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, 2002.

[298] S. K. Park and K. W. Miller. Random Number Generators: Good Ones Are Hard to Find. *Commun. ACM*, 31(10):1192–1201, 1988.

[299] M. Passing and F. Dressler. Experimental Performance Evaluation of Cryptographic Algorithms on Sensor Nodes. In *IEEE Int. Conf. on Mobile Ad Hoc and Sensor Systems (MASS'06)*, pages 882–887. IEEE, 2006.

[300] N. Patnaik, J. Hallett, and A. Rashid. Usability Smells: An Analysis of Developers' Struggle With Crypto Libraries. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 245–257, Washington, D.C., Aug. 2019. USENIX Association.

[301] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298, IETF, June 2011. URL `https://doi.org/10.17487/RFC6298`.

[302] B. Pearson, L. Luo, Y. Zhang, R. Dey, Z. Ling, M. Bassiouni, and X. Fu. On Misconception of Hardware and Cost in IoT Security and Privacy. In *53rd International Conference on Communications (ICC '19)*, pages 1–7, Piscataway, NJ, USA, 2019. IEEE.

[303] H. Petersen, P. Kietzmann, C. Gündogan, T. C. Schmidt, and M. Wählisch. Bluetooth Mesh under the Microscope: How much ICN is Inside? In *Proc. of 6th ACM Conference on Information-Centric Networking (ICN)*, pages 134–140, New York, 2019. ACM. URL `https://doi.org/10.1145/3357150.3357398`.

[304] H. Petersen, P. Kietzmann, T. C. Schmidt, and M. Wählisch. NDN meets BLE: A Transparent Gateway for Opening NDN-over-BLE Networks to your Smartphone. In *Proc. of 6th ACM Conference on Information-Centric Networking (ICN), Demo Session*, pages 175–176, New York, September 2019. ACM. URL `https://doi.org/10.1145/3357150.3357411`.

[305] C. Pham. Investigating and experimenting CSMA channel access mechanisms for LoRa IoT networks. In *Wireless Communications and Networking Conference (WCNC '18)*, pages 1–6, Piscataway, NJ, USA, 2018. IEEE.

[306] J. M. Pollard. Monte Carlo methods for index computation (mod $p$). *Mathematics of Computation*, 32(143):918–924, july 1978.

[307] G. C. Polyzos and N. Fotiou. Building a reliable Internet of Things using Information-Centric Networking. *Journal of Reliable Intelligent Environments*, 1(1):47–58, 2015.

[308] M. A. Prada-Delgado, A. Vázquez-Reyes, and I. Baturone. Trustworthy firmware update for Internet-of-Thing Devices using physical unclonable functions. In *Global Internet of Things Summit (GIoTS '17)*, pages 1–5, Piscataway, NJ, USA, 2017. IEEE.

[309] A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini. Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing. In *44th European Solid State Circuits Conference (ESSCIRC'18)*, pages 274–277, Piscataway, NJ, USA, 2018. IEEE.

[310] *zero-riscy: User Manual*. PULP Platform, January 2018. URL `https://www.pulp-platform.org/docs/user_manual.pdf`. Rev. 0.2.

[311] *RI5CY: User Manual*. PULP Platform, April 2019. URL `https://www.pulp-platform.org/docs/ri5cy_user_manual.pdf`. Rev. 4.0.

[312] M. T. Rahman, A. Hosey, Z. Guo, J. Carroll, D. Forte, and M. Tehranipoor. Systematic Correlation and Cell Neighborhood Analysis of SRAM PUF for Robust and Unique Key Generation. *Journal of Hardware and Systems Security*, 1(2):137–155, 2017.

[313] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, IETF, January 2012. URL `https://doi.org/10.17487/RFC6347`.

[314] T. Ristenpart and S. Yilek. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Network and Distributed System Security Symposium (NDSS '10)*, Reston, VA, USA, 2010. Internet Society.

[315] M. Rizzi, P. Ferrari, A. Flammini, E. Sisinni, and M. Gidlund. Using LoRa for industrial wireless networks. In *13th International Workshop on Factory Communication Systems (WFCS'17)*, pages 1–4, Piscataway, NJ, USA, 2017. IEEE Press.

[316] D. Ron, C.-J. Lee, K. Lee, H.-H. Choi, and J.-R. Lee. Performance Analysis and Optimization of Downlink Transmission in LoRaWAN Class B Mode. *IEEE Internet of Things Journal*, 7(8):7836–7847, 2020.

[317] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *IEEE Symposium on Security and Privacy (SP)*, pages 195–212, Piscataway, NJ, USA, 2017. IEEE Press.

[318] M. Rottleuthner, T. C. Schmidt, and M. Wählisch. Eco: A Hardware-Software Co-Design for In Situ Power Measurement on Low-end IoT Systems. In *ACM SenSys, 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys 2019)*, pages 22–28, New York, November 2019. ACM. URL `https://doi.org/10.1145/3362053.3363495`.

[319] V. Rožić and I. Verbauwhede. Hardware-Efficient Post-Processing Architectures for True Random Number Generators. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(7):1242–1246, 2019.

[320] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Proc. of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 237–249, New York, NY, USA, 2010. ACM.

[321] M. Saelens, J. Hoebeke, A. Shahid, and E. D. Poorter. Impact of EU duty cycle and transmission power limitations for sub-GHz LPWAN SRDs: an overview and future challenges. *EURASIP Journal on Wireless Communications and Networking*, 2019(219):219–251, 2019.

[322] M. Saito and M. Matsumot. Tiny Mersenne Twister (TinyMT): A small-sized variant of Mersenne Twister, 2011. URL `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT`. Retrieved 2019-11-24.

[323] M. G. Samaila, M. Neto, D. A. B. Fernandes, M. M. Freire, and P. R. M. Inácio. Challenges of securing Internet of Things devices: A survey. *Security and Privacy*, 1(2):e20, 2018.

[324] S. Satpathy, S. K. Mathew, V. Suresh, M. A. Anders, H. Kaul, A. Agarwal, S. K. Hsu, G. Chen, R. K. Krishnamurthy, and V. K. De. A 4-fJ/b Delay-Hardened Physically Unclonable Function Circuit With Selective Bit Destabilization in 14-nm Trigate CMOS. *IEEE Journal of Solid-State Circuits*, 52(4):940–949, 2017.

[325] D. Saxena, V. Raychoudhury, and N. SriMahathi. SmartHealth-NDNoT: Named Data Network of Things for Healthcare Services. In *Proc. of Workshop on Pervasive Wireless Healthcare (MobileHealth)*, pages 45–50, New York, NY, USA, 2015. ACM.

[326] A. Schaller. *Lightweight Protocols and Applications for Memory-Based Intrinsic Physically Unclonable Functions Found on Commercial Off-The-Shelf Devices*. Doctoral dissertation, Department of Computer Science, Technische Universität Darmstadt, December 2017. URL `http://tuprints.ulb.tu-darmstadt.de/7014/`.

[327] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, and G. Carle. Towards an Ecosystem for Reproducible Research in Computer Networking. In *Proc. of ACM SIGCOMM Reproducibility Workshop*, pages 5–8, New York, NY, USA, August 2017. ACM.

[328] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *27th Intern. Symp. on Power and Timing Modeling, Optimization and Simulation (PATMOS'17)*, pages 1–8, Piscataway, NJ, USA, 2017. IEEE.

[329] T. Schläpfer and A. Rüst. Security on IoT Devices with Secure Elements. Technical report, WEKA, 2019.

[330] T. C. Schmidt, S. Wölke, N. Berg, and M. Wählisch. Let's Collect Names: How PANINI Limits FIB Tables in Name Based Routing. In *Proc. of 15th IFIP Networking Conference*, pages 458–466, Piscataway, NJ, USA, May 2016. IEEE Press.

[331] E. M. Schooler, D. Zage, J. Sedayao, H. Moustafa, A. Brown, and M. Ambrosin. An Architectural Vision for a Data-Centric IoT: Rethinking Things, Trust and Clouds. In *IEEE 37th Intern. Conference on Distributed Computing Systems (ICDCS)*, pages 1717–1728, Piscataway, NJ, USA, June 2017. IEEE.

[332] G.-J. Schrijen and V. van der Leest. Comparative analysis of SRAM memories used as PUF primitives. In *DATE '12: Design, Automation Test in Europe Conference Exhibition*, pages 1319–1324, Piscataway, NJ, USA, 2012. IEEE.

[333] S. Schulz, A.-R. Sadeghi, and C. Wachsmann. Short Paper: Lightweight Remote Attestation Using Physical Functions. In *Proc. of the 4th ACM Conference on Wireless Network Security (WiSec '11)*, pages 109–114, New York, NY, USA, 2011. ACM.

[334] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050, IETF, November 2007. URL `https://doi.org/10.17487/RFC5050`.

[335] G. Selander, J. Mattsson, F. Palombini, and L. Seitz. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613, IETF, July 2019. URL `https://doi.org/10.17487/RFC8613`.

[336] A. Shamir. On the Generation of Cryptographically Strong Pseudorandom Sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

[337] W. Shang, Y. Yu, T. Liang, B. Zhang, and L. Zhang. NDN-ACE: Access Control for Constrained Environments over Named Data Networking. Technical Report NDN-0036, NDN, December 2015.

[338] W. Shang, A. Afanasyev, and L. Zhang. The Design and Implementation of the NDN Protocol Stack for RIOT-OS. In *Proc. of IEEE GLOBECOM 2016*, pages 1–6, Washington, DC, USA, 2016. IEEE.

[339] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang. Named Data Networking of Things (Invited Paper). In *Proc. of IEEE International Conf. on Internet-of-Things Design and Implementation (IoTDI)*, pages 117–128, Los Alamitos, CA, USA, 2016. IEEE Computer Society.

[340] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 623–656, July/Oct. 1948.

[341] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, IETF, June 2014. URL `https://doi.org/10.17487/RFC7252`.

[342] J. Shi and B. Zhang. NDNLP: A Link Protocol for NDN. NDN, Technical Report NDN-0006, NDN Team, July 2012.

[343] J. Shi, T. Liang, H. Wu, B. Liu, and B. Zhang. Ndn-nic: Name-based filtering on network interface card. In *Proc. of ACM ICN*, pages 40–49, New York, NY, USA, 2016. ACM.

[344] J. Shi, Y. Lu, and J. Zhang. Approximation Attacks on Strong PUFs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2138–2151, 2020.

[345] Y. Shiferaw, A. Arora, and F. Kuipers. LoRaWAN Class B Multicast Scalability. In *Proc. of 19th IFIP Networking Conference*, pages 609–613, Piscataway, NJ, USA, June 2020. IEEE Press.

[346] D. Shumow and N. Ferguson. On the Possibility of a Back Door in the NIST SP800-90 Dual EC PRNG. `http://rump2007.cr.yp.to/15-shumow.pdf`, last accessed 07-17-2020, 2007.

[347] T. Silde. Comparative Study of ECC Libraries for Embedded Devices. `https://tjerandsilde.no/files/Comparative-Study-of-ECC-Libraries-for-Embedded-Devices.pdf`, last accessed 10-09-2020, 2019.

[348] V. A. Siris, C. N. Ververidis, G. C. Polyzos, and K. P. Liolis. Information-Centric Networking (ICN) architectures for integration of satellites into the Future Internet. In *First AESS European Conference on Satellite Telecommunications (ESTEL'12)*, Piscataway, NJ, USA, 2012. IEEE.

[349] M. Slabicki, G. Premsankar, and M. Di Francesco. Adaptive Configuration of LoRa Networks for Dense IoT Deployments. In *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'18)*, pages 1–9, Piscataway, NJ, USA, 2018. IEEE Press.

[350] O. SpecWorks. Lightweight Machine to Machine Technical Specification: Core v1.2. Technical report, Open Mobile Alliance, 2020.

[351] W. Stallings. *Cryptography and Network Security.* Prentice Hall, Upper Saddle River, NJ, USA, 6 edition, 2014.

[352] A. Stanford-Clark and H. L. Truong. MQTT For Sensor Networks (MQTT-SN) Version 1.2. Protocol specification, IBM, November 2013. URL `http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf`.

[353] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The First Collision for Full SHA-1. In *Advances in Cryptology (CRYPTO '17)*, pages 570–596, Cham, Switzerland, 2017. Springer.

[354] *STM32F410 advanced Arm-based 32-bit MCUs.* STMicroelectronics, November 2018. Rev. 3.

[355] *Introduction to STM32 microcontrollers security.* STMicroelectronics, October 2019. Rev. 3.

[356] E. G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70:806–808, 1964.

[357] E. Strieder, C. Frisch, and M. Pehl. Machine Learning of Physical Unclonable Functions using Helper Data: Revealing a Pitfall in the Fuzzy Commitment Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES '21)*, 2021(2):1–36, 2021.

[358] S. S. Subramanian, J. Pasquale, and G. C. Polyzos. CoAP for Content-Centric Networks. In *Proc. of IEEE CCNC*, pages 467–472, Piscataway, NJ, USA, 2017. IEEE.

[359] M. Sýs and Z. Říha. Faster Randomness Testing with the NIST Statistical Test Suite. In *SPACE '14: Security, Privacy, and Applied Cryptography Engineering*, pages 272–284, Cham, Switzerland, 2014. Springer.

[360] S. Taneja, V. K. Rajanna, and M. Alioto. In-Memory Unified TRNG and Multi-Bit PUF for Ubiquitous Hardware Security. *IEEE Journal of Solid-State Circuits*, 57(1):153–166, 2022.

[361] T. Teubler, M. Hail, and H. Hellbrück. Efficient Data Aggregation with CCNx in Wireless Sensor Networks. In *19th Open European Summer School (EUNICE)*, volume 8115 of *LNCS*, pages 209–220, Berlin Heidelberg, 2013. Springer.

[362] *CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee/ZigBee IP Applications*. Texas Instruments, May 2013. Version C.

[363] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan. Performance evaluation of MQTT and CoAP via a common middleware. In *Proc. of ISSNIP*, pages 1–6, Piscataway, NJ, USA, 2014. IEEE.

[364] The Hacker News. Millions of IoT Devices Using Same Hard-Coded CRYPTO Keys. `https://thehackernews.com/2015/11/iot-device-crypto-keys.html`, last accessed 02-12-2022, 2015.

[365] The Hacker News. A Critical Random Number Generator Flaw Affects Billions of IoT Devices. `https://thehackernews.com/2021/08/a-critical-random-number-generator-flaw.html`, last accessed 29-03-2022, 2021.

[366] The Linux Kernel Development Community. Kconfig Language. `https://www.kernel.org/doc/html/latest/kbuild/kconfig-language.html`, last accessed 28-09-2020, 2020.

[367] S. Thielemans, M. Bezunartea, and K. Steenhaut. Establishing transparent IPv6 communication on LoRa based low power wide area networks (LPWANS). In *Wireless Telecommunications Symposium (WTS '17)*, pages 1–6, Piscataway, NJ, USA, 2017. IEEE.

[368] J. Tillmanns, J. Classen, F. Rohrbach, and M. Hollick. Firmware Insider: Bluetooth Randomness is Mostly Random. In *14th USENIX Workshop on Offensive Technologies (WOOT '20)*, Berkeley, CA, USA, 2020. USENIX Association.

[369] E. Tromer, D. A. Osvik, and S. Adi. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[370] B. Tsao, Y. Liu, and B. Dezfouli. Analysis of the Duration and Energy Consumption of AES Algorithms on a Contiki-Based IoT Device. In *16th Proc. of the EAI Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '19)*, pages 483–491, New York, NY, USA, 2019. ACM.

[371] H. Tschofenig and T. Fossati. Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. RFC 7925, IETF, July 2016. URL `https://doi.org/10.17487/RFC7925`.

[372] C. Tschudin, C. Scherb, et al. CCN Lite: Lightweight implementation of the Content Centric Networking protocol, 2018. URL `http://ccn-lite.net`.

[373] M. S. Turan, E. B. Barker, J. M. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle. Recommendation for the Entropy Sources Used for Random Bit Generation. Special Publication NIST SP 800-90B, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2018.

[374] V. van der Leest, B. Preneel, and E. van der Sluis. Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems (CHES '12)*, pages 268–282, Berlin, Heidelberg, 2012. Springer–Verlag.

[375] V. van der Leest, E. van der Sluis, G.-J. Schrijen, PimTuyls, and H. Handschuh. *Efficient Implementation of True Random Number Generator Based on SRAM PUFs*, pages 300–318. Springer, Berlin, Heidelberg, 2012.

[376] A. van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede. Secure PRNG Seeding on Commercial Off-the-shelf Microcontrollers. In *3rd International Workshop on Trustworthy Embedded Devices (TrustED '13)*, pages 55–64, New York, NY, USA, 2013. ACM.

[377] A. Varga. The OMNeT++ Discrete Event Simulation System, 2003.

[378] S. Vigna. An Experimental Exploration of Marsaglia's Xorshift Generators, Scrambled. *ACM Trans. Math. Softw.*, 26(4):59–82, 2016.

[379] S. Vigna. Further Scramblings of Marsaglia's Xorshift Generators. *J. Comput. Appl. Math.*, 315(C):175–181, 2017.

[380] X. Vilajosana, K. Pister, and T. Watteyne. Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration. RFC 8180, IETF, May 2017. URL `https://doi.org/10.17487/RFC8180`.

[381] B. C. Villaverde, D. Pesch, R. de Paz Alberola, S. Fedor, and M. Boubekeur. Constrained Application Protocol for Low Power Embedded Networks: A Survey. In *Proc. of 6th International Conf on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 702–707, Washington, DC, USA, 2012. IEEE Computer Society.

[382] V. D. Vincenzo, M. Heusse, and B. Tourancheau. Improving Downlink Scalability in LoRaWAN. In *IIEEE International Conference on Communications (ICC'19)*, Piscataway, NJ, USA, 2019. IEEE.

[383] C. Vinschen and J. Johnston. Newlib C library. `https://sourceware.org/newlib/`, last accessed 12-10-2020, 2020.

[384] J. von Neumann. Various Techniques Used in Connection with Random Digits. *J. Res. Nat. Bur. Stand. Appl. Math. Series*, 5:768–770, 1951.

[385] M. Vučinić, G. Selander, J. P. Mattsson, and D. Garcia-Carillo. Requirements for a Lightweight AKE for OSCORE. Internet-Draft – work in progress 04, IETF, June 2020. URL `https://datatracker.ietf.org/doc/html/draft-ietf-lake-reqs-04`.

[386] M. Wählisch, T. C. Schmidt, and M. Vahlenkamp. Bulk of Interest: Performance Measurement of Content-Centric Routing. In *Proc. of ACM SIGCOMM, Poster Session*, pages 99–100, New York, August 2012. ACM. URL `http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p99.pdf`.

[387] M. Wählisch, T. C. Schmidt, and M. Vahlenkamp. Backscatter from the Data Plane – Threats to Stability and Security in Information-Centric Network Infrastructure. *Computer Networks*, 57(16):3192–3206, Nov. 2013. URL `http://doi.org/10.1016/j.comnet.2013.07.009`.

[388] J. Walker. A Pseudorandom Number Sequence Test Program. `http://www.fourmilab.ch/random/`, last accessed 07-17-2020, 2008.

[389] L. Wang, A. Afanasyev, R. Kuntz, R. Vuyyuru, R. Wakikawa, and L. Zhang. Rapid Traffic Information Dissemination Using Named Data. In *Proc. of 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications (NoM)*, pages 7–12, New York, NY, USA, 2012. ACM.

[390] Q. Wang and J. Jiang. Comparative Examination on Architecture and Protocol of Industrial Wireless Sensor Network Standards. *IEEE Communications Surveys Tutorials*, 18 (3):2197–2219, 2016.

[391] Q. Wang, X. Vilajosana, and T. Watteyne. 6TiSCH Operation Sublayer (6top) Protocol (6P). RFC 8480, IETF, November 2018. URL `https://doi.org/10.17487/RFC8480`.

[392] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In *CRYPTO '05; Proceedings of the 25th Annual International Conference on Advances in Cryptology*, pages 17–36, Berlin, Heidelberg, 2005. Springer-Verlag.

[393] X. Wang, M. Magno, L. Cavigelli, and L. Benini. FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things. *IEEE Internet of Things Journal*, 7(5):4403–4417, 2020.

[394] T. Watteyne, M. Palattella, and L. Grieco. Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement. RFC 7554, IETF, May 2015. URL `https://doi.org/10.17487/RFC7554`.

[395] R. Whittle. Park-Miller-Carta Pseudo-Random Number Generator, 2005. URL `http://www.firstpr.com.au/dsp/rand31/`. Retrieved 2019-11-25.

[396] F. Wilde. Large Scale Characterization of SRAM on Infineon XMC Microcontrollers as PUF. In *Proc. of the 4th Workshop on Cryptography and Security in Computing Systems (CS2'17)*, pages 13–18, New York, NY, USA, 2017. ACM.

[397] F. Wilde and M. Pehl. On the Confidence in Bit-Alias Measurement of Physical Unclonable Functions. In *International New Circuits and Systems Conference (NEWCAS'19)*, Piscataway, NJ, USA, 2019. IEEE.

[398] F. Wilde, C. Frisch, and M. Pehl. Efficient Bound for Conditional Min-Entropy of Physical Unclonable Functions Beyond IID. In *International Workshop on Information Forensics and Security (WIFS'19)*, Piscataway, NJ, USA, 2019. IEEE.

[399] 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. RFC 6550, IETF, March 2012. URL `https://doi.org/10.17487/RFC6550`.

[400] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang. Neural Network Modeling Attacks on Arbiter-PUF-Based Designs. *IEEE Transactions on Information Forensics and Security*, 2022.

[401] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele. Increasing the Reliability of Wireless Sensor Networks with a Distributed Testing Framework. In *4th WS on Embedded Networked Sensors*, EmNets'07, pages 93–97. ACM, 2007.

[402] wolfSSL Inc. wolfCrypt Embedded Crypto Engine. `https://www.wolfssl.com/products/wolfcrypt/`, last accessed 10-11-2020, 2020.

[403] J. Woodage and D. Shumow. An Analysis of NIST SP 800-90A. In *EUROCRYPT '19: Advances in Cryptology*, volume 11477 of *LNCS*, pages 151–180, Berlin, Heidelberg, 2019. Springer.

[404] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor. Bit selection algorithm suitable for high-volume production of SRAM-PUF. In *International Symposium on Hardware-Oriented Security and Trust (HOST'14)*, pages 101–106, Piscataway, NJ, USA, 2014. IEEE.

[405] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys and Tutorials*, 16(2):1024–1049, 2014.

[406] G. Yapar, T. Tugcu, and O. Ermis. Time-Slotted ALOHA-based LoRaWAN Scheduling with Aggregated Acknowledgement Approach. In *25th Conference of Open Innovations Association (FRUCT'19)*, pages 383–390, Piscataway, NJ, USA, 2019. IEEE.

[407] M.-D. Yu and S. Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Design & Test of Computers*, 27(1):48–65, 2010.

[408] M.-D. Yu, R. Sowell, A. Singh, D. M'Raïhi, and S. Devadas. Performance metrics and empirical results of a PUF cryptographic key generation ASIC. In *International Symposium on Hardware-Oriented Security and Trust (HOST'12)*, pages 108–115, Piscataway, NJ, USA, 2012. IEEE.

[409] M.-D. M. Yu, D. M'Raïhi, S. Devadas, and I. Verbauwhede. Security and Reliability Properties of Syndrome Coding Techniques Used in PUF Key Generation, 2013.

[410] T. Yu, Z. Zhang, X. Ma, P. Moll, and L. Zhang. A Pub/Sub API for NDN-Lite with Built-in Security. Technical Report NDN-0071, NDN, Jan. 2021.

[411] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi. Remanence Decay Side-Channel: The PUF Case. *IEEE Transactions on Information Forensics and Security*, 11(6):1106–1116, 2016.

[412] Zephyr Project. Zephyr. `https://www.zephyrproject.org`, last accessed 07-17-2020, 2020.

[413] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73, 2014.

[414] M. Zhang, V. Lehman, and L. Wang. Scalable name-based data synchronization for named data networking. In *IEEE INFOCOM'17*, INFOCOM '17, pages 1–9, Los Alamitos, CA, USA, 2017. IEEE Computer Society.

[415] B. Zhou, M. Egele, and A. Joshi. High-performance low-energy implementation of cryptographic algorithms on a programmable SoC for IoT devices. In *High Performance Extreme Computing Conference (HPEC'17)*, pages 1–6. IEEE, 2017.

[416] L. Zhou, C. Su, Z. Hu, S. Lee, and H. Seo. Lightweight Implementations of NIST P-256 and SM2 ECC on 8-bit Resource-Constraint Embedded Device. *ACM Trans. Embed. Comput. Syst.*, 18(3):1–13, 2019.

[417] ZigBee Alliance. ZigBee Specification. Specification Document 05-3474-21, ZigBee Alliance, August 2015. URL `https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf`.

[418] D. Zorbas, K. Abdelfadeel, P. Kotzanikolaou, and D. Pesch. TS-LoRa: Time-slotted LoRaWAN for the Industrial Internet of Things. *Computer Communications*, 153:1–10, 2020.