# JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference

Markus Schuß[†], Carlo Alberto Boano[†], Manuel Weber[†], Matthias Schulz[‡], Matthias Hollick[‡], and Kay Römer[†]

[†]Institute for Technical Informatics, Graz University of Technology, Austria
[‡]Secure Mobile Networking Lab, Darmstadt University of Technology, Germany
{markus.schuss,cboano,manuel.weber,roemer}@tugraz.at ; {mschulz,mhollick}@seemoo.tu-darmstadt.de

## Abstract

Evaluating the performance of low-power wireless protocols in noisy environments in a repeatable and fully-automated way is still an open problem in our community. On the one hand, there is a lack of tools enabling the controllable and repeatable generation of interference using Wi-Fi devices. On the other hand, existing testbeds do not offer the automated generation of Wi-Fi interference on a large-scale. In this work, we present JamLab-NG, an open-source framework allowing the generation of controllable Wi-Fi interference using off-the-shelf devices such as the Raspberry Pi 3. JamLab-NG enables the fine-grained control of individual link-layer transmissions, avoiding the uncontrollable delays introduced by the network stack, the operating system, and the clear channel assessment procedure. Furthermore, JamLab-NG allows to generate repeatable Wi-Fi interference patterns by controlling radio settings such as the transmission speed and the packet length, which would otherwise be automatically adapted by the radio firmware at run-time.
We use JamLab-NG to augment a testbed and embed the generation of Wi-Fi interference into its automated execution of experiments. Among others, we allow remote configuration of the interference generated by individual Wi-Fi devices, and show that they can operate in a synchronized fashion. Finally, we use the augmented testbed to benchmark the performance of state-of-the-art IoT protocols under Wi-Fi interference in a repeatable and fully-automated way.

## Categories and Subject Descriptors

B8.2 [**Performance Analysis and Reliability**]

## General Terms

Design, Measurement, Performance, Reliability.

*Keywords*

Competition, Dependability, Performance, Testbeds.

## 1 Introduction

The prolific nature of Wi-Fi, combined with its high data rates and transmission power, as well as its large bandwidth, can be highly detrimental to the communications of co-located IoT devices using the 2.4 GHz ISM band. In the presence of Wi-Fi transmissions, for example, systems based on IEEE 802.15.4 or Bluetooth Low Energy typically experience an increased packet loss and number of retransmissions [22, 32]. This affects the latency, throughput, and energy efficiency of the network, which may be critical for IoT systems used in safety-critical application domains such as smart production and smart grids.

For this reason, over the past decade, both industry and academia have worked relentlessly in order to design new low-power wireless protocols that can sustain a reliable performance also in noisy environments [19, 21, 42, 43]. These new-generation protocols have proven that techniques such as constructive interference, flooding, and frequency-hopping can help in mitigating cross-technology interference [41]. However, how to compare the performance of these and other IoT protocols under Wi-Fi interference in an automated and repeatable way remains an open problem.

**Challenges.** This problem is mainly due to (i) the limitations of mote-based approaches commonly used to test protocols under interference, (ii) the inability of fully controlling the interference generated by actual Wi-Fi devices, and (iii) the lack of large-scale IoT testbeds offering the automated generation of Wi-Fi interference on a large scale.

*Limitations of mote-based approaches.* Several researchers rely on JamLab [9], a tool to produce repeatable interference using off-the-shelf motes [12, 19, 26, 43]. JamLab allows to use a fraction of the IEEE 802.15.4 nodes in a testbed to generate specific interference patterns by simply reprogramming the nodes intended as jammers with the proper software. However, Jamlab's ability to reproduce Wi-Fi traffic is limited, due to the limited bandwidth (3 MHz), transmission power (1 mW), and rate (250 kbps) of IEEE 802.15.4 transceivers. Indeed, JamLab can only emulate IEEE 802.11b traffic, as the limited throughput of motes based on IEEE 802.15.4 radios does not allow to emulate faster IEEE 802.11g/n transmissions. Furthermore, to emulate a Wi-Fi device using 20 or 40 MHz bandwidth channels, one would need to synchronize 4 to 8 JamLab nodes, respectively, hence using a large fraction of the nodes in a

International Conference on Embedded Wireless Systems and Networks (EWSN) 2019
25–27 February, Beijing, China

ISBN: 978-0-9949886-3-8

testbed. The number of necessary JamLab nodes would increase even further if one needs to cover the area that would be interfered by a Wi-Fi device transmitting at 100 mW using IEEE 802.15.4-based motes.

*Inability of fully controlling Wi-Fi transmissions.* Several works make use of actual Wi-Fi devices to test the performance of IoT protocols under interference [13, 24, 31, 44]. Whilst this approach is obviously superior to the use of motes (in the sense that the limitations on bandwidth, transmission power, and rate are intrinsically solved), it still cannot guarantee the repeatability of an experiment. A Wi-Fi pattern can indeed only be repeated when having full control of the individual link-layer transmissions. Approaches relying on a Wi-Fi device downloading large files from an access point [13, 20, 25], or generating a bandwidth-limited data transfer using applications such as `iperf` [18, 31, 33, 44] are unable to fully control the timing and properties of Wi-Fi transmissions. The problem lies in the inability of applications to directly access the radio firmware (which is typically closed-source). On the one hand, due to the radio's automatic adaptation of the transmission rate and packet size, an application lacks the means to control the channel occupancy in a fine-grained way. On the other hand, applications do not have control over the delays introduced by the network stack, the operating system (OS), and the actual radio firmware, which limits the control over the timing of individual link-layer transmissions. On top of this, Wi-Fi radios always perform a clear channel assessment (CCA) prior transmission and back-off when the channel is busy, which exacerbates the problem even further. As a result, as we experimentally show in Sect. 2, the repeatability of experiments carried out with off-the-shelf Wi-Fi devices is rather limited.

*Automation of large-scale experiments.* Having many Wi-Fi devices generate interference over a large area is a complex and time-consuming procedure. Wi-Fi devices require indeed the prior establishment of a connection to communicate. In conventional settings, Wi-Fi clients connect to an access point (AP) using a specific channel. However, when using *several* APs to generate interference over a large area, one needs to change their settings on a per-experiment basis, which is a huge configuration burden and is often unfeasible. In principle, Wi-Fi clients could be configured to act as *ad-hoc* APs. However, to avoid interfering with each other, multiple Wi-Fi clients cannot operate on adjacent channels. This makes the channel selection over large areas problematic, as it requires knowledge about the placement of each device and the propagation of its signals. For this reason, most researchers rely on temporary, static, and small-scale setups involving only one or at most few APs [1, 22, 32]. To tackle these limitations, we hence need a client-only solution, which does not rely on an active connection. Doing so with software-defined radios (SDRs) [17, 30] does not scale, due to the high costs of the required hardware. As a result, the automated generation of Wi-Fi interference in large-scale IoT testbeds remains an open challenge.

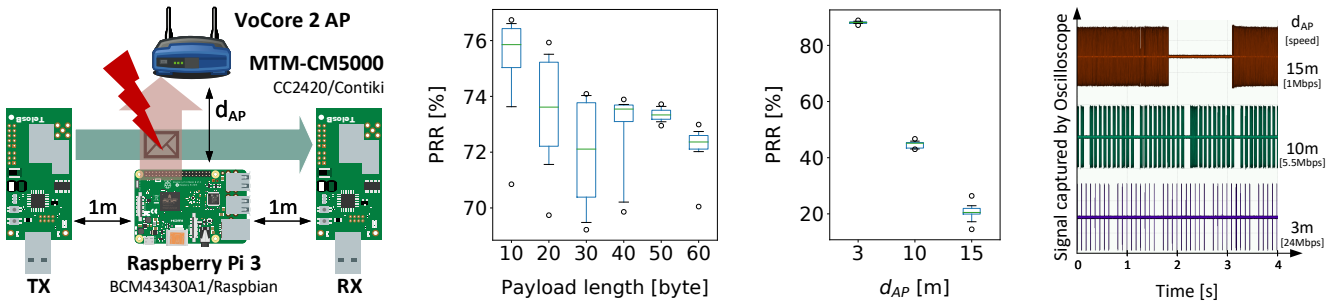
This state of affairs represents a serious problem, because it limits the possibility of *rigorously benchmarking* the dependability of low-power wireless protocols and IoT systems in the presence of Wi-Fi interference [7]. On the one hand, because of the limited repeatability and reproducibility of experiments, the obtained results can hardly be compared and generalized. On the other hand, the lack of automated testbed solutions, together with the high effort and costs in manually setting up experiments, discourages most researchers, who rather resort to the use of background Wi-Fi noise from offices where testbeds are deployed to test their protocols' performance [16, 46]. This practice, however, does not ensure repeatable and comparable experiments, and is therefore not suitable to benchmark the performance of IoT systems.

**Contributions.** To address all the aforementioned challenges, we present JamLab-NG, a framework that allows to generate controllable and repeatable Wi-Fi interference using low-cost off-the-shelf hardware. JamLab-NG is able to fully control the relevant physical layer settings of the Wi-Fi radio, as well as to schedule individual link-layer packets. This is achieved by modifying the Wi-Fi radio's firmware and by controlling it without the need to rely upon the network stack or the operating system (Sect. 3). In particular, JamLab-NG (i) allows to control parameters such as transmission rate, power, and packet size, (ii) supports disabling of the CCA procedure, and (iii) implements accurate timers in the Wi-Fi radio firmware that enable a fine-grained control of the timing of individual link-layer transmissions. JamLab-NG further simplifies the description of an interference pattern by allowing a developer to directly specify the physical properties of the interference to be generated (e.g., channel occupancy and burstiness), hence enabling the creation of a library of patterns to be used for benchmarking. We implement JamLab-NG on top of Nexmon [39], a popular C-based patching framework for Wi-Fi radios, and showcase the repeatability of the generated interference using the off-the-shelf Raspberry Pi 3 (Sect. 4). We further make JamLab-NG open-source[1] and keep its design modular: this, together with the extensive support of Nexmon for Broadcom (now Cypress) chipsets [28], enables an easy migration of JamLab-NG to other popular low-cost devices (Sect. 5). We then show how to integrate JamLab-NG's functionality into large-scale IoT testbed facilities by augmenting D-Cube [41], i.e., by adding the generation of Wi-Fi interference into its automated execution of experiments (Sect. 6). In particular, we allow the configuration of Wi-Fi interference via a REST API or Web interface, and show that each Wi-Fi device can autonomously control interference in a synchronized fashion without the need of additional infrastructure. We finally make use of the D-Cube testbed augmented with JamLab-NG to benchmark the performance of several IoT protocols under Wi-Fi interference, and show that their performance can be quantitatively compared in a repeatable, reproducible, and fully-automated way (Sect. 7).

## 2 Issues in Controlling Wi-Fi Transmissions

We start our discussion by experimentally showing the limitations of existing approaches making use of Wi-Fi devices to generate interference in terms of *repeatability* and *reproducibility*. The repeatability of an experiment can be defined as the variability of the measurements obtained when

[1] `http://www.iti.tugraz.at/JamLab-NG`




(a) Experimental setup (b) Repeatability problem (c) Reproducibility problem (d) RF signal at varying $d_{AP}$

**Figure 1. The inability of fully controlling the timing of Wi-Fi transmissions leads to a repeatability and reproducibility problem. The generated interference causes the PRR of communicating IEEE 802.15.4 nodes to largely vary across the same experiment (b) and across experiments with different $d_{AP}$ (c), due to delays introduced by operating system and network stack as well as due to the Wi-Fi radio's dynamic adaptation of parameters such as the transmission rate (d).**

repeating the same experiment multiple times. The reproducibility refers to the variability of the measurements obtained when replicating the same experiment in another setting, which hinders a consistent comparison of results [31].

**Experimental Setup.** We replicate a setup similar to the ones commonly used in the literature to artificially create noisy environments by generating Wi-Fi traffic with a given bandwidth [31, 33, 44]. As shown in Fig. 1, we let a pair of TelosB replicas (Advanticsys MTM-CM5000 embedding a TI CC2420 radio) exchange packets periodically in the presence of a Raspberry Pi 3 (RPi3) connected to a VoCore2 Wi-Fi AP. The RPi3 makes use of its embedded `bcm43430a1` radio chip to connect to the AP using Wi-Fi channel 6, and generates traffic using `ncat` in combination with `pv`: this allows to continuously transfer data from the RPi3 to the AP using TCP with a limited bandwidth $B_W$. Note that the AP is isolated from other networks and that all experiments are carried out in absence of external interference. The two TelosB nodes (TX and RX) are placed at 1 m distance from the RPi3, whilst the distance between the RPi3 and the AP varies across our experiments and it is referred to as $d_{AP}$. TelosB nodes make use of Contiki with `nullmac` and `nullrdc` to let the TX node periodically transmit 16 packets/sec to the RX node on channel 18, without the randomness or variability introduced by duty-cycling and MAC layers. We measure the packet reception ratio (PRR) as the ratio between the number of correctly-received packets and the number originally transmitted by the TX node. Each experiment stops after transmitting 10 000 packets, and is repeated ten times.

**Limited repeatability.** In a first set of experiments, we set $B_W$=500 kbps, and keep $d_{AP}$=0.5 m to ensure a stable connection, as well as to avoid that the two Wi-Fi devices backoff due to concurrent TelosB transmissions [22]. As both the IEEE 802.15.4 and the Wi-Fi network exchange messages at a constant rate, we expect the measured PRR to be highly repeatable over long runs, and to linearly decrease when using larger payloads, as shown in [9, 29]. Fig. 1(b) shows the results of 10 experiments: the green line indicates the median value, each box depicts the first and third quartile, with the whiskers showing the 5–95 percentile, and the dots indicating outliers. As one can observe, the PRR exhibits a large variance (up to 5%) despite the constant rate of both IEEE 802.15.4 and Wi-Fi traffic, revealing a severe problem in terms of repeatability. While the amount of data transmitted in both networks remains unchanged over time, and the timing of the IEEE 802.15.4 transmissions is tightly controlled by Contiki, the same does not hold true for the Wi-Fi network. We have indeed observed that the timing of the generated Wi-Fi transmissions significantly varies over time, due to the activities of the RPi3's OS (kernel and network stack) and radio module. Applications such as `ncat` and `iperf`, unfortunately, cannot control those activities in detail and hence cannot schedule precisely-timed packet transmissions. As our results show, this limits the repeatability of the generated interference, and also implies a problem in terms of reproducibility, as we illustrate next.

**Limited reproducibility.** In a second set of experiments, we set $B_W$=200 kbps and vary $d_{AP}$, emulating the different distances between a Wi-Fi node and an AP on a large-scale testbed. In principle, one would expect the communicating IEEE 802.15.4 nodes to sustain a higher PRR, as the distance $d_{AP}$ between RPi3 and AP increases. Indeed, whilst the transmissions of the RPi3 should affect the IEEE 802.15.4 communications in the same way (the distance between RPi3 and TelosB nodes does not change), the signal strength of the AP's transmissions would weaken with distance and may no longer jam IEEE 802.15.4 traffic. Instead, as shown in Fig. 1(c), the measured PRR exhibits the exact opposite trend, i.e., it significantly decreases when the two Wi-Fi devices are far away. The reason for this behavior can be explained by looking at Fig. 1(d), which shows an excerpt of the RF signal captured using a Keysight MSO-S 254A oscilloscope with a bandwidth of 2.5 GHz. The RPi3's radio *automatically adapts* the transmission rate in relation to the quality of the wireless link with the AP, resulting in a transmission rate of 1, 5.5, and 24 Mbps at 3, 10, and 15 m, respectively. This changes unpredictably the channel occupancy of the generated Wi-Fi traffic, significantly affecting the way ongoing IEEE 802.15.4 transmissions are disturbed, and impairing the reproducibility of experiments.

In summary, our experiments highlight that (i) the reliance on (and impact of) existing Wi-Fi infrastructure (APs), together with (ii) the inability to control the physical settings of the Wi-Fi radio as well as the OS' activities limit the control

over the timing of individual link-layer transmissions – impairing the reproducibility and repeatability of experiments.

# 3 JamLab-NG: Design and Implementation

We next present the design of JamLab-NG: a framework that allows to generate repeatable and reproducible interference using common Wi-Fi devices. After clarifying the design rationale (Sect. 3.1), we present JamLab-NG's architecture (Sect. 3.2) and discuss the functionality and implementation of its core modules (Sect. 3.3 to 3.5).

## 3.1 Design Rationale

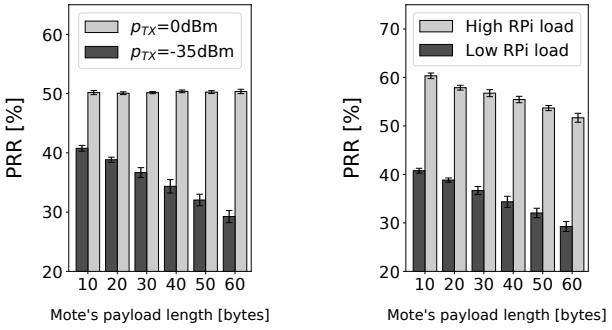
To mitigate the challenges shown in Sect. 2, one first needs to gain access to the radio's physical layer (PHY) and the ability of sending data without an access point.

**Using monitor mode.** Several chipsets offer the ability to work in monitor mode, which allows a device to send data without the need to associate to an AP. Although not supported by every Wi-Fi card, this mode allows applications like *scapy* [35] to send arbitrary, raw frames without the need of an established connection or additional infrastructure. Moreover, when using monitor mode, one can also configure PHY settings such as channel and transmission rate: this is necessary, as the radio is no longer connected to an AP, and hence unable to determine these settings autonomously.

While the use of monitor mode seems to be the solution to many of the challenges highlighted in Sect. 2, its use to send individual frames does still not allow to avoid the overhead as well as the uncontrollable delays introduced by the OS and the radio firmware. In addition, even on devices where monitor mode is available, several low-level settings such as the possibility to disable the CCA, the spacing between packets, or the per-packet transmission rate cannot be modified.

The ability to control these settings in a fine-grained way is very important: for example, depending on the strength of the received signal, a Wi-Fi radio with CCA enabled backs-off and delays its transmissions [22]. This means that the timing and the amount of the generated Wi-Fi interference becomes dependent on external influences such as the transmissions and settings of other nearby devices (e.g., employed channel, transmission power and rate). We show this by reusing the same setup described in Sect. 2 to let the RPi3 in monitor mode inject 763 bytes-long frames every 13 ms at 1 Mbps using *scapy*. Fig. 2(a) shows that the impact of the generated Wi-Fi interference on the PRR of the two IEEE 802.15.4 nodes varies as a function of the TX mote's transmission power $P_{TX}$ (0 or -35 dBm). Similar effects would occur if the motes would change their relative position, transmission rate, or channel. Gaining control of the radio's PHY settings and the ability to modify them is hence key to enable reproducible and repeatable experiments.

**Altering the radio firmware.** To modify the inner workings of a Wi-Fi radio, one can alter its firmware, i.e., the actual software running on the module. The source code of some Wi-Fi cards has been made available by its vendor, e.g., Atheros USB cards based on the `ath9k_htc` chip. The availability of such an open-source firmware has been used, among others, to build makeshift spectrum analyzers [18], or to enable energy detection at runtime and enable cross-technology communication [3, 15]. Open-source firmwares




(a) Effect of Wi-Fi's CCA (b) Effect of the RPi3's CPU load

**Figure 2. Limitations in the use of Wi-Fi's monitor mode when not being able to disable CCA (a) and bypass the OS (b): the impact of interference can be vastly different.**

have also been modified to enable even lower-level access to the internals of the Wi-Fi card. For example, Vanhoef et al. have disabled the CCA and removed the interframe gap on `ath9k_htc` chips in order to build a *reactive jammer* generating traffic as soon as a Wi-Fi transmission was detected on the air [49]. Other radios for which the source code was not made available by their vendors have been reverse-engineered, e.g., the popular Broadcom (now Cypress) `bcm43` series [37]. This enabled the use of monitor mode, normally unavailable on these cards, and has also been used to disable CCA and implement reactive jamming [36]. JamLab-NG exploits a combination of monitor mode with low-level access to the radio firmware to transmit repeatable interference patterns without the need to connect to an AP.

**Bypassing the OS.** This combination, however, may still be insufficient to generate repeatable interference on low-end devices. For example, when triggering the transmission of individual Wi-Fi packets from an userland application, the timing of the resulting interference is – among others – strongly affected by the load of the host CPU. Fig. 2(b) shows what happens when repeating the same experiment shown in Fig. 2(a) while the CPU of the RPi3 is under a constant artificial 100% load. The light grey bars show that the PRR sustained by the two motes is significantly higher ($\approx$20%) when the RPi3 is running with a high CPU load. The dark bars in Fig. 2 are measured with the constant 60% CPU load introduced by *scapy*. This shows that the userland application triggering the generation of interference should *not* handle individual packets, but only signal the beginning and the end of the interference process. Every other operation (e.g., loading and scheduling of packets) needs to be handled by the radio itself. We do so by creating a tool that triggers the generation of interference from within the radio module[2]. This way, by avoiding that the application triggers the transmission of individual frames, we minimize the number of calls through the OS, and avert uncontrollable delays.

**Describing the interference properties.** When using the radio module to handle the generation of interference upon

[2]Note that also reactive jamming approaches do not rely on an userland application scheduling individual packets, but instead trigger the transmission of frames upon detection of Wi-Fi activity [36, 49]. This, however, does not allow to generate interference at arbitrary points in time.

a trigger from an userland application, we would need to store long sequences of packets or embed lengthy descriptions about the timing of packet transmissions inside the radio firmware. However, as the latter has a very constrained memory (in the order of a few kB), this is not possible.

To overcome this constraint, we need to make use of a high-level description of interference without having to explicitly specify the characteristics and payload of the individual packets to be sent. We hence derive a lightweight model of Wi-Fi traffic and use it to generate controllable interference from the radio module despite its memory constraints.

**Parametrizing the interference properties.** Following this rationale, the userland application can trigger the generation of pre-defined interference models with a single call through the OS. To avoid the generation of fixed models, we still need to allow an application to parametrize the model, so to vary the characteristics of interference arbitrarily at runtime.

## 3.2 JamLab-NG's Architecture

We design JamLab-NG in order to enable Wi-Fi devices supporting monitor mode to generate controllable and repeatable interference. Monitor mode is supported by nearly every vendor, including Atheros (now Qualcomm), Intel, as well as RaLink, and is also available on reverse-engineered Broadcom (now Cypress) chips using Nexmon [37].

JamLab-NG encompasses two tools named `Jelly` and `Confiture`. With `Jelly`, one has the ability to generate interference by directly sending packets to the Wi-Fi radio in monitor mode. Although this does not solve the problem of bypassing the OS (see Sect. 3.1), this is the only way that the few Wi-Fi radios, for which no open-source or reverse-engineered firmware exists, have to generate controllable interference. All other Wi-Fi radios, instead, can make use of `Confiture`: a solution that bypasses the OS by embedding the interference generation directly into the radio firmware.

`Confiture` is split between a userland application (app) and a scheduler running directly on the Wi-Fi radio. The app is used to pass the properties of the interference to be generated to the scheduler in the form of *model parameters* (no individual packets). The scheduler is used to autonomously trigger the transmission of packets according to these model parameters. This allows us to trigger the generation of interference from the `Confiture` app using a single call, hence bypassing the OS and avoiding uncontrollable delays.

The key difference between the two tools is that `Confiture` can be used for repeatable interference generation from a Wi-Fi radio while the device performs other activities, such as recording the serial output, observing GPIO pins, or performing energy measurements. In contrast, `Jelly` requires the dedicated use of the Wi-Fi device for interference generation, and no other tasks should run on the OS in order to ensure repeatability. This makes `Confiture` suitable to extend common IoT testbeds [10, 23, 41, 45], which typically carry out all these activities, with highly-repeatable interference generation, as discussed in Sect. 6.

Fig. 3 shows an overview of the resulting architecture of JamLab-NG. We base our illustration on our reference implementation for the off-the-shelf RPi3 embedding a `bcm43` Wi-Fi chip. As there is no open-source implementation of

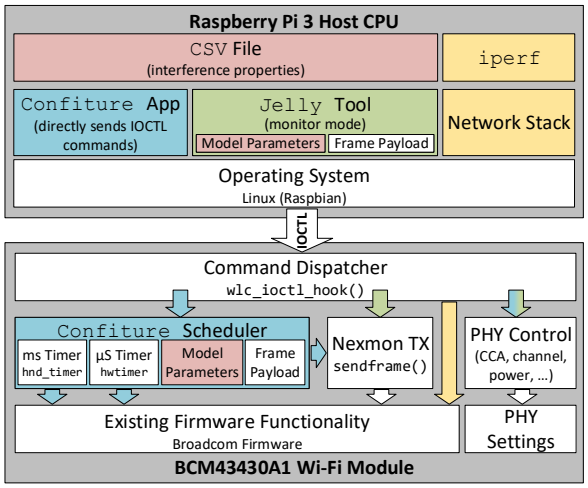



**Figure 3. A sketch of JamLab-NG's architecture.**

the `bcm43` firmware, we make use of Nexmon [37], a C-based patching framework for a wide range of Broadcom radios. This enables us to extend the Wi-Fi firmware with new functions without breaking existing functionality. A Wi-Fi device can hence normally connect to APs and a user can still run tools like `iperf` without any modification.

Both `Confiture` and `Jelly` read a common comma separated value (`CSV`) file containing the properties of the interference to be generated. This simplifies the description of an interference pattern by allowing a non-expert to directly specify interference characteristics such as channel occupancy and burstiness, hence enabling the creation of a library of patterns to be used for benchmarking. In particular, the `CSV` file contains in each line a list of model parameters that are used by `Jelly` and by `Confiture`'s scheduler to configure the transmission rate, as well as the length and timing of individual frames. Each line of the `CSV` also contains a timestamp to allow the model parameters to be changed over time and generate different interference patterns at runtime.

Differently from `Jelly`, the `Confiture` tool only sends commands to the radio using directly the input/output control (`IOCTL`) interface. The latter defines high-level commands that are identified by an integer number. `Confiture` makes use of these commands to change the model parameters stored in the scheduler, or to start/stop the interference generation, hence bypassing the OS functionality to send packets. Both tools share (i) the use of `IOCTL` commands to modify PHY settings such as channel, power, and CCA, as well as (ii) Nexmon's `sendframe()` function to inject the frame payload into the existing firmware functionality.

The injection of frames is performed in both tools according to an *interference model*. While in `Confiture` this task is performed by a scheduler embedded in the radio, in `Jelly` this functionality is kept in the userland. `Confiture`'s scheduler and `Jelly` operate according to the same principle, and call Nexmon's `sendframe()` function depending on the timing information provided by the model parameters. We next describe how such model parameters can be derived (Sect. 3.3), as well as the implemention of `Jelly` and `Confiture` (Sect. 3.4 and 3.5).

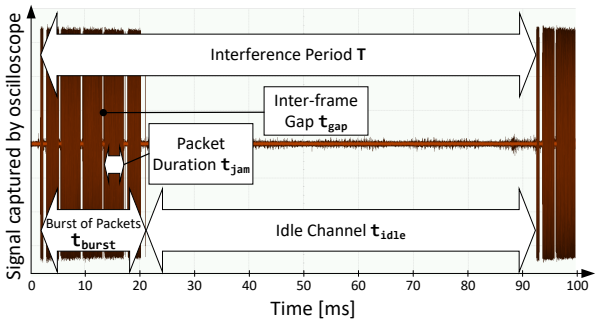



**Figure 4. Periodically repeating RF signal measured by a Keysight MSO-S 254A oscilloscope when transmitting bandwidth-limited TCP traffic (from Fig. 1(d)).**

## 3.3 Interference Model

The operation of `Confiture`'s scheduler and `Jelly` are based on a lightweight model specifying how often and for how long interference should be generated. To derive such a model, we analyze the characteristics of Wi-Fi traffic in the time domain using a mixed-signal oscilloscope when transmitting data with a given application. When making use of `ncat` and `pv` to produce bandwidth-limited TCP traffic as shown in Sect. 2, the resulting RF signal in the time domain follows a strongly-periodic pattern. For this type of periodic bandwidth-limited TCP traffic, we therefore create a model which generates periodic bursts of interference over time.

Fig. 4 shows a magnification of the bandwidth-limited TCP traffic depicted in Fig. 1(d). One can clearly identify a burst of Wi-Fi packets separated by a variable inter-frame gap, followed by a phase in which the channel remains idle. We can hence identify an interference period $T$ during which the channel is occupied by a burst of Wi-Fi packets ($t_{burst}$), followed by an idle phase ($t_{idle}$). The burst of packets is comprised of several frames, each of length $L$ bytes and duration $t_{jam}$. Consecutive frames are separated by a short idle phase due to the inter-frame gap ($t_{gap}$). The time each packet is on air depends heavily on the employed transmission rate $R$, which determines the amount of bits transmitted per second.

The duration in milliseconds of each individual packet $t_{jam}$ depends on the amount of data, as well as on the rate at which it is sent, and can be hence computed as:

$$t_{jam} = t_{preamble} + {}^{(L\cdot 8)}/_{R} \cdot 1000 \tag{1}$$

where $t_{preamble}$ is the time in milliseconds necessary to transmit the default, long IEEE 802.11 preamble and PLCP header at the fixed transmission rate of 1 Mbps:

$$t_{preamble} = \frac{144bit + 48bit}{1Mbps} \cdot 1000 \tag{2}$$

Using $t_{jam}$ and identifying $n$ as the number of consecutive packets sent within a burst, we can derive $t_{burst}$ in ms as:

$$t_{burst} = (t_{jam} \cdot n) + t_{gap} \cdot (n-1) \tag{3}$$

With this simple model, one can create a repeating signal emulating the traffic generated by a bandwidth-limited TCP connection. As we discuss in Sect. 5, one can enhance this model with controllable jitter, as well as derive other models for different types of Wi-Fi traffic.

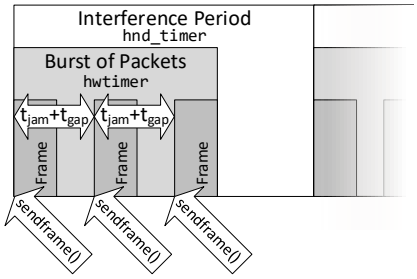



**Figure 5. `Confiture` uses two types of timers to transmit frames according to the model shown in Sect. 3.3.**

## 3.4 `Jelly`: Implementation

We base the design of `Jelly` on *scapy*, a powerful open-source packet manipulation program written in python. While originally designed to perform attacks such as ARP poisoning on Ethernet [35], it has recently gained the ability to read and modify `radiotap` headers. With this ability, it can be used to inject IEEE 802.11 frames and pass to the radio parameters such as the transmission rate, but also vendor specific extensions such as the ability to turn off CCA.

The main feature of *scapy* used in our design is its `L2socket`, which allows us to send raw IEEE 802.11 frames at the maximum rate supported by the Wi-Fi card. However, this is only possible with pre-built frame payloads, as the serialization of python frame objects to their binary representation requires a large amount of CPU resources – hence introducing large uncontrollable delays.

## 3.5 `Confiture`: Implementation

`Confiture` is split between an *app* that orchestrates the generation of repeatable interference and a *scheduler* that takes care of the actual transmission of link-layer frames.

**App.** The userland application reads from a `CSV` file (i) the relative time-stamp since the start of the interference generation, (ii) the model parameters, and (iii) the PHY settings to be employed (channel, power, CCA). It then issues via IOCTL commands the start and stop of the interference generation to the scheduler running in the radio module.

**Scheduler.** The scheduler's main task is to precisely schedule the transmissions of frames based on the interference model described in Sect. 3.3 and its parameters. To this end, the scheduler stores the model parameters in a dedicated memory area after receiving them from the app via `IOCTL` commands. The scheduler makes use of two types of timers: a high-resolution µs-timer (`hwtimer`) and a low-resolution ms-timer (`hnd_timer`). The high-resolution timer is used to model the interframe gap and send the individual packets, whilst the low-resolution timer represents the interference period $T$, as shown in Fig. 5. Every time the `hnd_timer` fires, the `hwtimer` fires multiple times in a row in order to send the individual packets using the `sendframe()` function. The latter takes as argument the transmission rate $R$, as well as a buffer with the payload to be sent. As this buffer is freed with every send operation, randomly generating a new payload each time would limit the rate at which packets can be sent. Therefore, we also store a predefined (maximum length) payload in the scheduler, and allocate a new buffer (in which we copy $L$ bytes) before calling `sendframe()`.

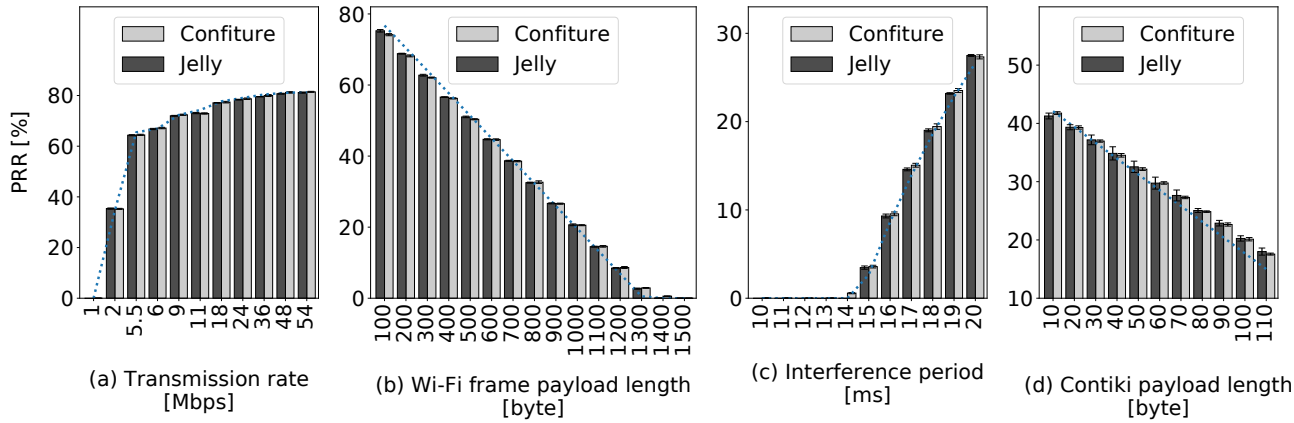



**Figure 6. PRR sustained by IEEE 802.15.4 nodes for `Confiture` (lighter grey bars) and `Jelly` (darker grey bars) when using different model parameters for JamLab-NG and when the motes employ a different payload length.**

## 4 Evaluating JamLab-NG's Repeatability

We evaluate the ability of JamLab-NG to generate controllable and repeatable Wi-Fi interference experimentally. In particular, our evaluation answers the following questions:

- How repeatable is the impact of JamLab-NG's interference when using different model parameters?
- Does the impact of JamLab-NG's generated interference follow the expected trends?
- Is JamLab-NG actually more repeatable than common bandwidth-limited Wi-Fi traffic generators?

We start by evaluating the impact of JamLab-NG's interference on the performance of IEEE 802.15.4 communications when using different model parameters (Sect. 4.1). We show that such an impact is highly repeatable and follows the expected trends, thanks to JamLab-NG's fine-grained control of interference. Thereafter, we show that the repeatability achieved by JamLab-NG is significantly higher than the one obtained using tools such as `iperf` and `ncat` (Sect. 4.2).

**Setup.** Our evaluation is based on the same experimental setup described in Sect. 2. The only difference is that the VoCore2 AP has now been removed: the RPi3 hence runs JamLab-NG and generates interference without the need of additional infrastructure. As in Sect. 2, we assess the impact of the generated interference by measuring the PRR of two motes placed in proximity of the RPi3. Unless stated differently, the TX mote makes use of packets with a 40-bytes payload sent every 62.5 ms on channel 26, whilst JamLab-NG employs Wi-Fi channel 14 and the following model parameters: $T$=13 ms, $R$=1 Mbps, $n$=1, and $L$=1526 bytes.

### 4.1 Impact of Model Parameters

We evaluate the repeatability of JamLab-NG by analyzing the variability in the PRR sustained by the MTM-CM5000 nodes over tens of experiments and hundreds of thousands transmissions. We use as metrics: (i) the average standard deviation of PRR across all experiments ($\sigma_{PRR}$), and (ii) the range of PRR values across all experiments ($\Theta_{PRR}$).

**Transmission rate.** We first analyze the impact of JamLab-NG's interference on the PRR of the communicating motes when using different transmission rates ($R$). The `bcm43430a1` can select several values for $R$ depending on the employed modulation, namely:

- DSSS: 1 and 2 Mbps;
- CCK: 5.5 and 11 Mbps;
- OFDM: 6, 9, 18, 24, 36, 48, and 54 Mbps.

Fig. 6(a) shows that the PRR of the two motes increases when using a higher transmission rate. This is expected, as higher transmission rates make use of shorter Wi-Fi frames, causing a lower channel utilization. The PRR does not change significantly at higher rates: this is due to the fixed (slow) preamble, whose impact becomes higher when $R$ increases. One can also observe that the use of $R$=1 Mbps results in a PRR of 0: at such low rate, when using $L$=1526 bytes, $t_{jam}$ is indeed almost equal to $T$ (12.5 and 13 ms respectively), which is insufficient to let Contiki successfully transmit an entire packet over-the-air without collisions.

Overall, JamLab-NG exhibits a very high repeatability, when using both `Jelly` (darker grey bars) and `Confiture` (lighter grey bars). The standard deviation of the PRR is in average only 0.17% for `Jelly` and 0.25% for `Confiture`, as highlighted in Table 1, whilst the PRR range is 0.59 and 0.78%, respectively. Note that `Jelly` exhibits a slightly higher repeatability than `Confiture` for two reasons. On the one hand, there is no other task running on the RPi3's CPU, which gives `Jelly` nearly full control of the 1.2 GHz quad-core. Second, `Jelly` and `Confiture` make use of different timers. Whilst Python's time module used by `Jelly` sustains the same standard deviation of the `hwtimer` (275 µs), `Confiture` also makes use of the `hnd_timer`, whose standard deviation is slightly higher (482 µs).

**Length of Wi-Fi frame payload.** We next analyze the impact on PRR when varying JamLab-NG's burst duration ($t_{jam}$). To this end, we vary the frame's payload length $L$ between 100 and 1500 bytes (note that $L$ does not include the length of the constant preamble). Fig. 6(b) shows that the measured PRR decreases linearly with the Wi-Fi frame lenght $L$, up to a point in which the channel is almost fully saturated ($L$=1500, resulting in a $t_{jam}$=12.3 ms).

Also in this case, JamLab-NG exhibits a high repeatability, both when using `Jelly` and `Confiture`. $\sigma_{PRR}$ is only

**Table 1. Average standard deviation of PRR ($\sigma_{PRR}$) and range of PRR values ($\Theta_{PRR}$) across all experiments for `Confiture` and `Jelly` when using different model parameters and when the motes employ different payload lengths.**

| JamLab-NG's Tool | `Confiture` | | | `Jelly` | | |
|---|---|---|---|---|---|---|
| | $\sigma_{PRR}$ (%) | $\Theta_{PRR}$ (%) | Model Error (%) | $\sigma_{PRR}$ (%) | $\Theta_{PRR}$ (%) | Model Error (%) |
| Transmission rate ($R$) | 0.25 | 0.78 | 0.43 | 0.17 | 0.59 | 0.61 |
| Length of Wi-Fi frame payload ($L$) | 0.19 | 0.62 | 1.18 | 0.16 | 0.52 | 0.88 |
| Interference period ($T$) | 0.14 | 0.41 | 1.33 | 0.09 | 0.41 | 1.33 |
| Contiki payload length ($L_{payload}$) | 0.23 | 0.75 | 1.26 | 0.70 | 2.18 | 1.50 |

0.16% for `Jelly` and 0.19% for `Confiture`, as highlighted in Table 1, whilst $\Theta_{PRR}$ is 0.52 and 0.62%, respectively.

**Interference period.** The impact of the interference period $T$ used by JamLab-NG on the PRR follows the expected trend: the higher $T$, the higher the PRR sustained by the motes. As we make use of $L$=1526 bytes and $R$=1 Mbps, for $T \leq 13$ ms, the channel is fully saturated and the two IEEE 802.15.4 nodes are unable to communicate. Fig. 6(c) shows that the repeatability of JamLab-NG is very high, with `Jelly` and `Confiture` sustaining a $\sigma_{PRR}$ of only 0.09 and 0.14%, respectively, as highlighted in Table 1.

**Length of IEEE 802.15.4 packets.** We finally show that the interference generated by JamLab-NG is also highly repeatable regardless the size of the packets transmitted by the two motes. As the maximum size of an IEEE 802.15.4 packet is 127 bytes, we subtract Contiki's Rime and MAC overhead and vary the payload between 10 and a maximum of 110 bytes. Fig. 6(d) shows that the longer the packets exchanged by the two motes, the lower the PRR. Whilst this is expected [9, 29], it is interesting to observe the small standard deviation and range of PRR values, especially when comparing them to the ones obtained in Fig. 1(b). As shown in Table 1, `Jelly` and `Confiture` sustain a $\sigma_{PRR}$ of 0.70 and 0.23%, as well as a $\Theta_{PRR}$ of 2.18 and 0.18%, respectively.

**Actual vs. expected impact of interference.** The four plots shown in Fig. 6 also embed a dotted blue line that indicates the expected PRR sustained by the two IEEE 802.15.4 nodes. We use this line to show that the impact of the interference produced by JamLab-NG is not arbitrary, but indeed follows the expected trends. The expected PRR is estimated based on the model parameters used by JamLab-NG, the time in ms necessary to transmit an IEEE 802.15.4 packet over the air ($t_{packet}$), and a simple model capturing the probability of a collision $P$. We compute $t_{packet}$ as:

$$t_{packet} = {}^{(L_{header}+L_{payload})\cdot 8}/_{R_{ieee}} \quad (4)$$

where $R_{ieee}$ is the transmission rate of an IEEE 802.15.4 radio (250 kbps), $L_{header}$ is the header length (17 bytes), and $L_{payload}$ is the length of the payload.

Following the mathematical model describing the likelihood that two periodically recurring events have to coincide by Richards [34], we derive:

$$P_0 = \frac{t_{jam} \cdot t_{packet}}{T \cdot T_{packet}} \qquad w = \frac{t_{jam} + t_{packet}}{T \cdot T_{packet}}$$
$$P = (P_0 + w * t) \cdot 100 \quad \text{for } t \leq Max(T, T_{packet}) \quad (5)$$

where $T$ and $t_{jam}$ are the interference period and burst duration used by JamLab-NG, respectively; and $T_{packet}$ is the rate at which the IEEE 802.15.4 nodes transmit packets (in our case 62.5 ms). Note that this simple model assumes (i) all collisions to lead to a packet loss, and (ii) no jitter in the timing of transmissions. Using $P$, the estimated PRR in % of the two motes when generating interference using JamLab-NG can be computed as $100 - P$.

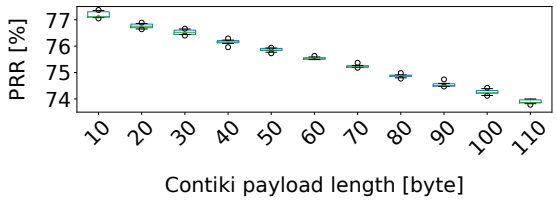



**Figure 7. Variability of the PRR sustained by the motes when using `Confiture` to model a rate-limited TCP connection. The repeatability is notably higher compared to the use of classic tools such as `ncat` (Fig. 1(b)).**

Fig. 6 shows that the estimated PRR (blue dotted line) follows accurately the actual interference generated by JamLab-NG. Table 1 also lists the average difference between the estimated PRR and the measured one. On average, the estimated PRR is always within 1.5% for both `Confiture` and `Jelly`, which shows that the impact of JamLab-NG's generated interference is highly deterministic and leads to the expected packet loss.

## 4.2 Comparison to Existing Tools

After showing the impact of the individual model parameters, we move back our focus to the original problem described in Sect. 2. Our goal is to compare the repeatability of the impact of JamLab-NG's interference with the one caused by existing tools generating a bandwidth-limited data transfer. To this end, we adjust JamLab-NG's model parameters to the ones used by `ncat` in the experiments shown in Fig. 1(b). We hence create longer interference bursts using $n$=32 packets of $L$=1526 bytes that are sent at a transmission rate $R$=54 Mbps every $T$=100 ms: this allows us to mimic a similar bandwidth-limited TCP connection to the AP with good signal quality.

Fig. 7 shows the impact of the interference generated by JamLab-NG (using `Confiture`) on the PRR of the two IEEE 802.15.4 nodes. When comparing Fig. 7 and Fig. 1(b), it is clear that JamLab-NG brings huge improvements in terms of repeatability, with significantly less outliers and a much smaller standard deviation. In Fig. 7, the PRR of the communicating motes exhibits a much clearer trend, with $\sigma_{PRR}$=0.08% and $\Theta_{PRR}$=0.25% only – compared to 1.41% and 4.11% obtained by `ncat` in Fig. 1(b), respectively.

## 5 Discussion

JamLab-NG is a complex framework whose features go well beyond the ones described in the reference design on Sect. 3. In this section, we aim to give a brief overview of the capabilities and functionalities that have already been added, or that can be integrated into JamLab-NG in the near future.

**Interference library.** In Sect. 3.3 we have described an exemplary model for bandwidth-limited traffic that can be used to emulate the behavior of applications like `iperf` and `ncat`. In principle, JamLab-NG's can be easily extended with other models using `CSV` files collected in a common library and easy to share. For example, one can implement probabilistic models similar to the ones supported by the original Jamlab [9]: `Confiture`'s implementation already includes a pseudo-random number generator with a configurable seed that can be used exactly for this purpose. Furthermore, one can also create arbitrary sequences of frames and share them as standard `PCAP` files. This can be done, for example, by recording Wi-Fi traffic using a radio sniffing packets in monitor mode. This functionality is already embedded in `Jelly`, but its implementation has been omitted from Sect. 3 due to space constraints.

**Porting JamLab-NG.** The `bcm43430a1` Wi-Fi chip embedded in the RPi3, on which our reference design is based, can also be found on other off-the-shelf boards such as the Raspberry Pi Zero W or the Banana Pi M2 Zero. As the radio chip runs the same firmware, these platforms *are all already supported* by JamLab-NG (both `Confiture` and `Jelly`). These boards, which do not have a dedicated Ethernet jack, but resemble instead an USB dongle, can be plugged into more powerful devices without Wi-Fi cards in order to extend them with the ability to generate repeatable interference (e.g., FlockLab's observer nodes [23]). Furthermore, as outlined in Sect. 3, `Confiture`'s design can also be ported to other cards with open-source firmware, such as platforms based on `ath9k_htc` chips. Using the available open-source implementation and the functionality documented in ModWiFi [49], this process should be straightforward.

**Observing interference.** The Broadcom `bcm43` chipset can also be used as RSSI sniffer. Unlike the Atheros `ath9k` chips, which only give the information whether the energy on the channel is above or below a given threshold, the `bcm43` can carry out a continuous energy detection and return numeric RSSI values. This feature could be used in the future to extend JamLab-NG with the ability to not only generate interference, but also to observe the amount of surrounding interference. This would allow testbeds to quantitatively measure if two experiments have been run under similar conditions and pave the way for solutions enabling an easier comparability of experimental results, a well-known problem in benchmarking low-power wireless systems [7].

**Advanced features.** JamLab-NG's reference design is based on the RPi3 platform. When using newer Wi-Fi radio chips, such as the `bcm43455c0` embedded in the RPI3's successor, the Raspberry Pi 3 Model B+, one can even implement more advanced forms of interference generation. For example, the undocumented registers available in the `bcm43455c0` allow to trigger the actuation of specific sub-carriers of the Wi-Fi band and can be exploited to send raw IQ signals directly [38]. This could be used, for example, to emulate the signals produced by other devices using the 2.4 GHz ISM band, e.g., IEEE 802.15.4 and Bluetooth Low Energy.

## 6 Augmenting IoT Testbeds with JamLab-NG

In the previous sections we have shown how JamLab-NG empowers the generation of repeatable interference using individual off-the-shelf Wi-Fi devices. Our ultimate goal, however, is its integration in IoT testbeds and its use for rigorous benchmarking of low-power wireless systems. We describe next the main challenges when integrating JamLab-NG into existing testbeds, discuss the necessary pre-requisites, and show a full-fledged integration into D-Cube [41].

**Pre-requisites.** In order to be extended with JamLab-NG, an IoT testbed obviously needs to be equipped with devices embedding a Wi-Fi radio. If this is not the case (e.g., FlockLab [23]), one can simply retrofit the testbed's *observer nodes*[3] with one of the many off-the-shelf USB Wi-Fi cards based on the `ath9k_htc`, or with a Raspberry Pi Zero W (as cheap as 10 EUR), as discussed in Sect. 5. If the existing observer nodes in a testbed already embed a Wi-Fi radio supporting monitor mode (e.g., w-iLab.t [18] and TWIST [48]), one can seamlessly make use of `Jelly`. The several IoT testbeds that are based on the Raspberry Pi 3 [10, 23, 41, 45] can support both `Jelly` and `Confiture` out of the box.

**Integration aspects.** If the observer nodes in a testbed meet the aforementioned pre-requisites, two main aspects need to be addressed when integrating JamLab-NG.

*Distribution of model parameters.* As a first step, one needs to distribute the model parameters used for interference generation in every observer node that should act as jammer (i.e., that should be used to generate Wi-Fi interference). As JamLab-NG makes use of a plain-text `CSV` file, the latter can either be copied to the various nodes using tools like `scp`, or be downloaded by each node from a REST API using `curl`.

*Synchronized operations.* As a second step, one needs to synchronize the activities of the different observer nodes acting as jammer. The simplest approach is to trigger the generation of interference separately from the experiment using the network interface (e.g., via `ssh` or using a daemon listening on a Web-socket connected to the server controlling the experiment). A more elegant solution consists in exploiting the capabilities that are already present in most testbeds, such as the automatic execution of experiments. JamLab-NG has actually been designed with this in mind, and embeds the ability to trigger the generation of interference using one of the GPIOs on the Wi-Fi device in use. One could hence connect a GPIO pin of each observer node to the reset pin of a target low-power sensor node (which is already toggled by the testbed infrastructure at the beginning of each experiment) and indirectly synchronize the operation of all jammers.

**Augmenting D-Cube.** D-Cube is a low-cost, open-source testbed that supports the remote, automatic execution of experiments using a Web interface or REST API [40, 41]. In

[3] We call observer nodes (or observers) the testbed devices used to reprogram the target low-power sensor nodes, to readout their serial output, as well as to measure their latency, energy consumption and GPIO activities.

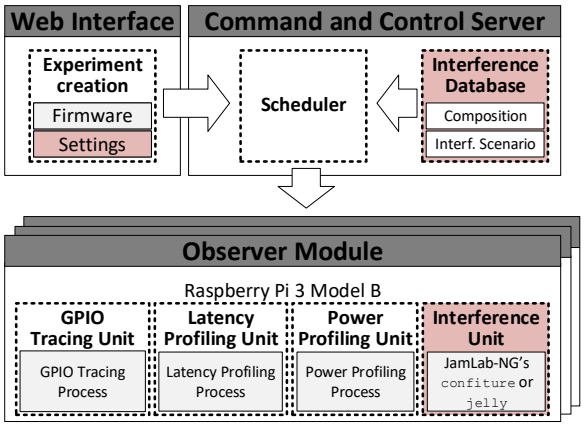



**Figure 8. Additional modules of D-Cube's architecture when augmented with JamLab-NG (marked in red).**

order to augment this testbed with JamLab-NG, we add three modules to its architecture (marked in red in Fig. 8).

*Interference unit.* On each observer node (RPi3), an interference unit runs in parallel to the GPIO tracing, latency profiling, and power profiling units. Once an experiment is scheduled to start, each observer node autonomously downloads its `CSV` file from a common interference database via a REST API.It then starts the `Confiture` app, which passes the model parameters from the `CSV` file to the scheduler running on the radio. Note that, as `Confiture` essentially requires no CPU resource, the operations of the other measurement units (GPIO, latency, power) are unaffected.

*Interference database.* In order to easily distribute the `CSV` files to the observer nodes and to keep a library of interference patterns, we have added an interference database on the server coordinating the execution of D-Cube's experiments. This interference database stores a list of interference *scenarios*, each of which is a direct representation of the `CSV` file, i.e., it embeds a series of timestamps and the configuration of interference (model parameters). By default, a scenario is applied to all observer nodes in the testbed. In case some observers need to interfere in a different way, one can associate in a *composition* table, a specific scenario to each of the RPi3 in the testbed, as shown in Fig. 9. Using the info in the composition table, a `CSV` file can be generated on the fly and autonomously downloaded by each RPi3.

*Experiment settings.* D-Cube allows the scheduling of new experiments using a Web interface or REST API. Users can already configure specific settings for each experiment, such as their duration and the logging of serial output. In addition to this, D-Cube now allows to select an optional interference pattern (`jamming_level`) to be generated during each experiment. Such `jamming_level` is associated with an entry in D-Cube's central database, which describes the model parameters to be used on each observer node in the testbed.

*Synchronizing D-Cube's operations.* To ensure that the operations of the observer nodes acting as jammer are synchronized to each other once an experiment is started, we rely on D-Cube's existing functionality to toggle the reset pin of each target low-power sensor node in the testbed. To this

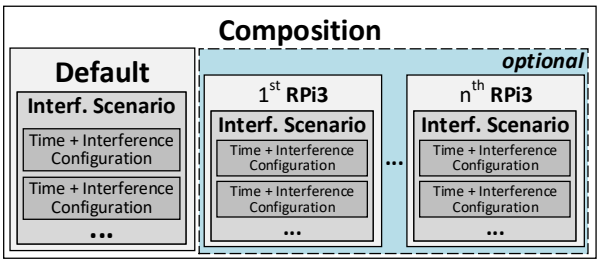



**Figure 9. Info stored in D-Cube's interference database.**

end, `Confiture` has been extended with the ability to observe a GPIO pin on the RPi3 and synchronize the start/stop of interference accordingly. In this way, no explicit communication between D-Cube and JamLab-NG is required. Several experiments have shown that the synchronization that can be reached in `Confiture` when exploiting the reset pin of the 51 MTM-CM5000 nodes in the D-Cube testbed exhibits a standard deviation of only 93 ms. This delay is due to the employed motes, and does not affect JamLab-NG's repeatability on a large scale, as shown in the next section.

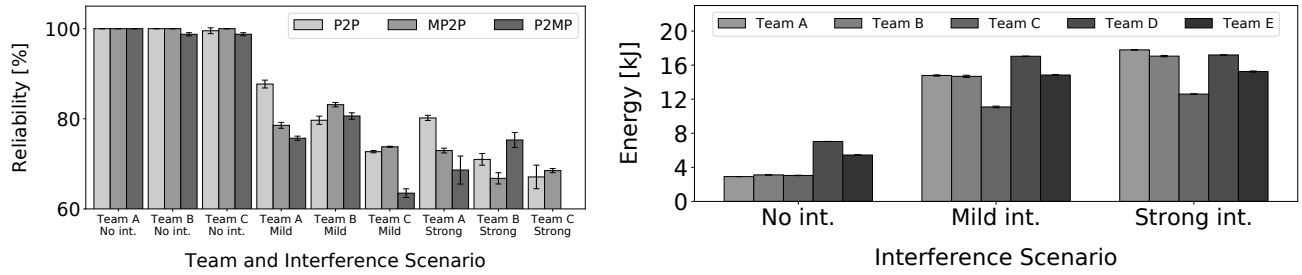
## 7 JamLab-NG in Action on D-Cube

We finally make use of the D-Cube testbed augmented with JamLab-NG to benchmark the performance of state-of-the-art IoT protocols under controllable and repeatable Wi-Fi interference. Towards this goal, we organized a public competition[4] and invited the authors of 10 influential IoT protocols to participate and compete with each other.

**Benchmarking scenario.** Differently from the measurements shown in Sect. 4, which were table experiments with only pairs of motes, we now make use of all 51 nodes available in D-Cube. During a preparation phase, we allowed all contestants to properly configure and optimize their protocols, such that they could obtain the best performance (reliability, timeliness, and energy-efficiency) despite the generated Wi-Fi interference. All contestants could already make use of the augmented D-Cube testbed during the preparation phase, and select the interference pattern to be generated by JamLab-NG (running `Confiture`) when creating an experiment using a Web interface or a REST API.

We provide seven different interference scenarios to `Confiture`, but we specifically focus on two in this paper, due to space constraints. The first scenario, named *mild* interference, emulates the operation of Wi-Fi devices in common office environments, with bursts of Wi-Fi activity followed by periods with an idle channel. The bursts of Wi-Fi activity had different characteristics over time (e.g., $L$ varied between 100 and 1500 bytes), and we also let each RPi3 change sporadically its channel (1–14), as well as its transmission power (0-200 mW). A second scenario, named *strong* interference, resembles the *mild* interference scenario, but all RPi3 nodes in the testbed make use of a transmission power of 250 mW (24 dBm). This *strong* scenario may be more challenging than what would be found in typical buildings and offices, but served to push the benchmarked protocols to the edge. Note that we also benchmark all protocols

[4] `https://iti-testbed.tugraz.at/blog/tag/ewsn2018/`




(a) Reliability of competing protocols

(b) Energy consumption of competing protocols

**Figure 10. Performance of different IoT protocols during a public competition aimed at benchmarking their performance under Wi-Fi interference. When using D-Cube augmented with JamLab-NG, the results are highly repeatable.**

in absence of any interference, in order to have a baseline.

We benchmark the performance of the 10 protocols using three types of traffic: (i) *point-to-point*: a single source transmitting to a single destination, (ii) *multipoint-to-point*: several sources to one destination, and (iii) *point-to-multipoint*: one source transmitting to multiple destinations. A detailed description of the evaluation scenario can be found in [40].

**Benchmarking results.** We discuss the results obtained by the different protocols during the competition by showcasing their repeatability. The actual performance of the 10 protocols (and the reasons behind it) would deserve a lengthy discussion, and is hence beyond the scope of this paper.

Fig. 10 shows the reliability (in terms of number of correctly-received information) and the energy consumption sustained by some of the competing protocols. Fig. 10(a) focuses on the reliability of three protocols making use of constructive interference: BigBangBus [14], Crystal [47], and Chaos [27] (we have masked the name of the protocols from the plot, as this paper does not focus on their relative performance). The PRR measured when running the protocols several times is highly repeatable, despite having 51 RPi3 nodes *concurrently* generating interference in our evaluation scenarios. In particular, the average standard deviation across all protocols was only 0.23, 0.71, and 2.03% with no, *mild* and *strong* interference, respectively. The average standard deviation hence grows with the intensity of the generated interference: this is due to the lower availability of the channel, which amplifies the variability of protocols' behavior. Note that all benchmarking activities were run at night, in order to avoid external sources of interference.

Fig. 10(b) shows the total energy consumed by all 51 nodes in the testbed measured using D-Cube's power profiling capability. The results are, also in this case, highly repeatable, with an average standard deviation of only 0.02, 0.07, and 0.04 kJ with no, *mild* and *strong* interference, respectively.

## 8 Related Work

Several researchers in the IoT community have dealt with the generation of interference, mostly in the context of evaluating protocol performance, or in order to evaluate the impact of jamming attacks on the security of wireless systems.

**Generating interference for protocol testing.** The ability to experiment in noisy environments on a large-scale while ensuring comparability of results has been a major challenge for the low-power wireless community [7]. A typical approach followed by several researchers is to rely on existing background Wi-Fi noise during daytime in offices or other locations in which testbeds are installed [8, 16, 46]. As this interference cannot be controlled, other works rely instead on a Wi-Fi device downloading large files from an access point [13, 20, 25], or generate a bandwidth-limited data transfer using applications such as `iperf` [18, 31, 33, 44].

As the coordination of multiple Wi-Fi devices is complex to manage on a large-scale (see Sect. 1), several researchers have resorted to the use of motes to generate interference [12, 19, 26, 43]. JamLab [9], for example, is a tool that allows off-the-shelf low-power motes to generate controllable and realistic interference. Although it simplifies the generation of interference on large-scale testbeds with minimal overhead, the interference generated using JamLab is subject to the limitations of IEEE 802.15.4 radios, and is not suitable to generate Wi-Fi interference on a large scale.

A few works use specialized HW such as SDRs to generate realistic interference [5, 11, 17]. This approach, however, relies on a powerful PC to precisely control the generated signals [6] and does not scale. Besides the large costs of SDR devices and the PCs steering their operations, when building large-scale SDR testbeds, one needs to rely on coaxial wires and a 1 PPS signal, which is impractical in most installations.

JamLab-NG was designed to remedy all the intrinsic limitations of the aforementioned approaches, as it enables the controllable generation of Wi-Fi interference on large-scale testbeds using low-cost off-the-shelf devices.

**Jamming attacks.** The ability to generate interference using Wi-Fi devices has also been used in a security context. The most common attack carried out using Wi-Fi interference is denial of service (DoS) [4, 49]. Packet injection is also typically used in this context to forge malicious (but valid) packets for de-authentication attacks [2] (management traffic) or key reuse [50]) to break Wi-Fi's encryption.

Unlike JamLab-NG, the goal of such attacks is not to generate a repeatable interference pattern, but to either completely deny access to the medium, to exploit weaknesses in the management traffic and force disconnection of clients (layer 2), or to break the connections' encryption and perform attacks on higher levels of the communication stack.

## 9 Conclusions and Future Work

In this paper, we have presented JamLab-NG, a framework that enables the creation of controllable and repeatable

Wi-Fi interference using low-cost hardware. After describing the design and implementation of its core components, we have experimentally shown how the impact of the interference generated by JamLab-NG is significantly more repeatable than the one observed using traditional systems. Furthermore, we have shown how to integrate JamLab-NG in IoT testbeds in order to evaluate protocol performance under Wi-Fi interference in a repeatable and fully-automated way. We believe that JamLab-NG paves the way for a rigorous benchmarking of IoT systems, and expect its open-source availability to facilitate the engagement of the low-power wireless community, enabling consistent comparisons.

## 10 Acknowledgments

This work was performed within the LEAD-Project "Dependable Internet of Things in Adverse Environments", funded by Graz University of Technology.

## 11 References

[1] M. D. Abrignani et al. Testing the Impact of Wi-Fi Interference on Zigbee Networks. In *Proc. of the EMTC Conf.*, 2014.

[2] J. Bellardo et al. 802.11 Denial-of-service Attacks: Real Vulnerabilities and Practical Solutions. In *Proc. of the* $12^th$ *SSYM Symp.*, 2003.

[3] A. Bereza et al. Cross-Technology Communication Between BLE and Wi-Fi Using Commodity Hardware. In *Proc. of the* $14^{th}$ *EWSN Conf., demo session*, 2017.

[4] K. Bicakci et al. Denial-of-Service Attacks and Countermeasures in IEEE 802.11 Wireless Networks. *Comp. Stand. & Interf.*, 31(5), 2009.

[5] B. Bloessl et al. An IEEE 802.11a/g/p OFDM Receiver for GNU Radio. In *Proceedings of the* $2^{nd}$ *SRIF Worksh.*, 2013.

[6] B. Bloessl, C. Leitner, F. Dressler, and C. Sommer. A GNU Radio-based IEEE 802.15.4 Testbed. In *Proc. of the* $12^{th}$ *FGSN Conf.*, 2013.

[7] C. A. Boano et al. Towards a Benchmark for Low-power Wireless Networking. In *Proc. of the* $1^{st}$ *CPSBench Worksh.*, 2018.

[8] C. A. Boano and K. Römer. External radio interference. In *Radio Link Quality Estimation in Low-Power Wireless Networks*, SpringerBriefs in Electrical and Computer Engineering - Cooperating Objects. 2013.

[9] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. A. Zúñiga. JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. In *Proc. of the* $10^{th}$ *IPSN Conf.*, 2011.

[10] Z. Brodard et al. Rover: Poor (but Elegant) Man's Testbed. In *Proc. of the* $13^{th}$ *PE-WASUN Symp.*, 2016.

[11] L. S. Cardoso et al. Reliable and Reproducible Radio Experiments in FIT/CorteXlab* SDR testbed: Initial Findings. In *Proc. of the* $12^{th}$ *CrownCom Conf.*, 2017.

[12] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy Links, Low Power, High Throughput. In *Proc. of the* $9^{th}$ *ACM SenSys Conf.*, 2011.

[13] P. Dutta et al. Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless. In *Proc. of the* $8^{th}$ *ACM SenSys Conf.*, 2010.

[14] A. Escobar et al. BigBangBus. In *Proc. of the* $15^{th}$ *EWSN Conf., competition session*, 2018.

[15] P. Gawłowicz and A. Zubow. Practical cross-technology radio resource management between LTE-U and WiFi. In *Proc. of the IEEE INFOCOM Conf., demo session*, 2018.

[16] O. Gnawali et al. Collection Tree Protocol. In *Proceedings of the* $7^{th}$ *ACM SenSys Conf.*, 2009.

[17] A. Hithnawi et al. Controlled Interference Generation for Wireless Coexistence Research. In *Proc. of the SRIF Worksh.*, 2015.

[18] iLab-t Testbeds' 1.0.0 Documentation. https://doc.ilabt.imec.be/ilabt-documentation/wilabfacility.html.

[19] T. Istomin et al. Interference-resilient Ultra-low Power Aperiodic Data Collection. In *Proc. of the* $17^{th}$ *IPSN Conf.*, 2018.

[20] M. S. Kang et al. Adaptive Interference-Aware Multi-Channel Clustering Algorithm in a ZigBee Network in the Presence of WLAN Interference. In *Proc. of the* $2^{nd}$ *ISWPC Symp.*, 2007.

[21] A. King, J. Brown, and U. Roedig. DCCA: Differentiating Clear Channel Assessment for Improved 802.11/802.15.4 Coexistence. In *Proc. of the* $10^{th}$ *WiMob Conf.*, 2014.

[22] C.-J. M. Liang et al. Surviving Wi-Fi Interference in Low Power ZigBee Networks. In *Proc. of the* $8^{th}$ *ACM SenSys Conf.*, 2010.

[23] R. Lim et al. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proc. of the* $12^{th}$ *IPSN Conf.*, 2013.

[24] S. Moeller et al. Routing without Routes: the Backpressure Collection Protocol. In *Proc. of the* $9^{th}$ *IPSN Conf.*, 2010.

[25] R. Musaloiu-E. and A. Terzis. Minimising the Effect of Wi-Fi Interference in 802.15.4 Wireless Sensor Networks. *IJSNet*, 3(1), 2007.

[26] B. A. Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-Power Listening Goes Multi-Channel. In *Proc. of the* $10^{th}$ *DCOSS Conf.*, 2014.

[27] B. A. Nahas and O. Landsiedel. Aggressive Synchronous Transmissions with In-network Processing for Dependable All-to-All Communication. In *Proc. of the* $15^{th}$ *EWSN Conf., competition session*, 2018.

[28] Nexmon GitHub repository. https://github.com/seemoo-lab/nexmon.

[29] C. Noda et al. On Packet Size and Error Correction Optim. in Low-Power Wireless Networks. In *Proc. of the* $10^{th}$ *SECON Conf.*, 2013.

[30] Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT). http://www.orbit-lab.org/.

[31] G. Z. Papadopoulos, A. Gallais, G. Schreiner, and T. Noël. Importance of Repeatable Setups for Reproducible Experimental Results in IoT. In *Proc. of the* $13^{th}$ *PE-WASUN Symp.*, 2016.

[32] M. Petrova, L. Wu, P. Mähönen, and J. Riihijärvi. Interference Measurements on Performance Degradation between Colocated IEEE 802.11g/n and 802.15.4 Networks. In *Proc. of the* $6^{th}$ *ICN Conf.*, 2007.

[33] S. Pollin et al. Harmful Coexistence Between 802.15.4 and 802.11: A Measurement-based Study. In *Proc. of the CrownCom Conf.*, 2008.

[34] P. I. Richards. Probability of Coincidence for Two Periodically Recurring Events. *The Annals of Mathematical Statistics*, 19(1), 1948.

[35] Scapy: Packet Crafting for Python2 and Python3. https://scapy.net/.

[36] M. Schulz et al. Massive Reactive Smartphone-based Jamming Using Arbitrary Waveforms and Adaptive Power Control. In *Proc. of the* $10^{th}$ *WiSec Conf.*, 2017.

[37] M. Schulz et al. Nexmon: Build Your Own Wi-Fi Testbeds with Low-Level MAC & PHY-Access using Firmware Patches on Off-the-Shelf Mobile Devices. In *Proc. of the* $11^{th}$ *WinTECH Worksh.*, 2017.

[38] M. Schulz et al. Teaching Smartphones to Transmit Raw Signals and to Extract Channel State Information to Implement Practical Covert Channels over Wi-Fi. In *Proc. of the* $16^{th}$ *MobiSys Conf.*, 2018.

[39] M. Schulz, D. Wegemer, and M. Hollick. The Nexmon Firmware Analysis and Modification Framework: Empowering Researchers to Enhance Wi-Fi Devices. *COMCOM*, 129(1), 2018.

[40] M. Schuß, C. A. Boano, and K. Röemer. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the* $1^{st}$ *CPSBench Worksh.*, 2018.

[41] M. Schuß, C. A. Boano, M. Weber, and K. Römer. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *Proc. of the* $14^{th}$ *EWSN Conf.*, 2017.

[42] M. Sha, G. Hackmann, and C. Lu. Energy-efficient Low Power Listening for Wireless Sensor Networks in Noisy Environments. In *Proc. of the* $12^{th}$ *IPSN Conf.*, 2013.

[43] J. Shi, M. Sha, and Z. Yang. DiGS: Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks. In *Proc. of the* $38^{th}$ *ICDCS Conf.*, 2018.

[44] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson. EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. of the* $12^{th}$ *MobiHoc Conf.*, 2011.

[45] The Binghamton University Wireless Embedded System Testbed. http://www.cs.binghamton.edu/ msha/testbed.

[46] V. Toldov et al. Experimental Evaluation of Interference Impact on the Energy Consumption in Wireless Sensor Networks. In *Proc. of the* $17^{th}$ *WoWMoM Symp.*, 2016.

[47] M. Trobinger et al. CRYSTAL Clear: Making Interference Transparent. In *Proc. of the* $15^{th}$ *EWSN Conf., competition session*, 2018.

[48] TWIST Testbeds' 1.12.0 Documentation. https://www.twist.tu-berlin.de/testbeds/wireless.html#twist-nucs.

[49] M. Vanhoef and F. Piessens. Advanced Wi-Fi Attacks Using Commodity Hardware. In *Proc. of the* $30^{th}$ *ACSAC Conf.*, 2014.

[50] M. Vanhoef and F. Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proc. of the ACM CCS Conf.*, 2017.